الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

BADJI MOKHTAR ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR ANNABA

جامعة باجي مختار - عنابة

Année 2017

Faculté des sciences de l'ingéniorat
Département d'électronique

THÈSE

Présentée en vue de l'obtention du diplôme de DOCTORAT DE TROISIEME CYCLE

LES TÉLÉCOMMUNICATIONS SANS FIL ET LES RÉSEAUX DE COMMUNICATION EMBARQUÉS

Option : Télécommunications

Par :

BENAICHA Ramzi

Directeur de thèse :

Pr. TAIBI **Mahmoud** **Université d'Annaba**

DEVANT LE JURY

PRESIDENT:

Pr. TOUMI **Salah** **Université d'Annaba**

EXAMINATEURS:

Pr. TALEB-Ahmed **Abdelmalik** **Univ-Valenciennes France**

Pr. BARTIL **Arres** **Université de Sétif**

Dr. SAOUCHI **Kaddour** **Université d'Annaba**

REMERCIEMENTS

- Louange à Allah, Seigneur de l'univers, le tout Puissant, le Miséricordieux, qui me protège, m'inspire et me comble de ses bienfaits. Je lui en rends toujours grâce.

- La réalisation de cette thèse a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma reconnaissance.

- Je voudrais tout d'abord adresser toute ma gratitude au directeur de cette thèse **TAIBI MAHMOUD**, pour les conseils judicieux et constants qu'il m'a prodigués tout au long de ce travail, sa grande disponibilité ainsi que la confiance qu'il m'a toujours inspirée. Il m'a été d'un soutien scientifique et moral tout simplement inestimable. Qu'il soit assuré de ma reconnaissance, mon respect et mon amitié indéfectibles.

- Mes remerciements s'adressent à l'ensemble des membres de mon jury d'avoir accepté de juger et d'évaluer ces travaux de thèse. Je remercie sincèrement **TOUMI SALAH**, Professeur à l'université d'Annaba, de m'avoir fait l'honneur de présider mon jury ; qu'il trouve ici l'expression de mon profond respect. Je remercie également **BARTIL ARRES**, Professeur à l'université de Sétif, **TALEB-AHMED. A**, Professeur à l'université de Valenciennes (France) ainsi que **SAOUCHI KADDOUR**, Maître de Conférences à l'université d'Annaba d'avoir accepté de me donner un peu de leurs précieux temps pour faire part de mon jury en tant qu'examinateurs.

- Je tiens également à exprimer mes remerciements et ma reconnaissance :

A monsieur K. BEY Professeur à l'université d'Annaba pour son aide et son encouragement précieux.

A Messieurs Y. DIAF et R. TOUMI S. BOUDALIA pour leurs aides et leurs encouragements.

A tous les membres du laboratoire LERICA que j'ai eu le plaisir de côtoyer durant mes longues années de thèse.

DÉDICACE

C'est avec profonde gratitude et sincères mots que je dédie ce travail à :

Mes chers parents,

Que nulle dédicace ne puisse exprimer mes sincères sentiments, pour leur patience illimitée, leur encouragement continu, leur aide, en témoignage de mon profond amour et respect pour leurs grands sacrifices.

Ma chère femme « INSAF »,

Qui m'a aidé et supporté dans les moments difficiles, que tous les mots ne sauraient exprimer son amour et son affection inconditionnels durant toutes ces longues années.

Mon enfant « IYAD »,

Qui est ma source d'inspiration et mon plus grand soutien.

Mes chers frères et sœur, « RAOUF et sa petite famille », « NASREDDINE » et « SOUHIR et sa petite famille »,

Pour leur grand amour et leur soutien, qu'ils trouvent ici l'expression de ma haute gratitude.

Mes beaux- parents, mes belles sœurs « MERIEM » et « LINA » et mon beau frère « RAOUF »,

Qui m'ont toujours poussé et motivé dans mes études.

Mes chers amis, « HANI », « REDA » et « MOHAMED »,

Pour leur affection, compréhension et patience.

Tous ceux qui ont une relation de proche ou de loin avec la réalisation de cette thèse.

الملخص

مع تطور شبكة الإنترنت، نحن اليوم نشهد انفجار حركة المرور عبر الشبكات. هذا الانفجار من الحركة ليس من دون عواقب. مشكلتين لا بد من ايجاد الحلول لهما، المشكلة الأولى هي قدرة الروابط التي تحمل معلومات، والآخرة طريقة التوجيه للعثور على معظم مسارات التكلفة. اليوم الحل موجود لهما جزئيا حيث نجد مسافة خوارزميات النواقل وحالة ارتباط المعلومات. ولكن بروتوكولات التوجيه الحالية لا تأخذ في الاعتبار سوى القليل من جودة الخدمة، ويمكن أن تتكيف بسهولة مع بيئة الديناميكية التي تتميز بها قمم الحمل الوجيز والمكثف. الهدف من التوجيه هو تحديد مسار أو مجموعة من الروابط ضمن شروط معينة، لتأسيس اتصال مع عقدة المصدر وعقدة الوجهة. والغرض من خوارزمية التوجيه هو السماح لحساب الطريق بين العقدتين بمعيار معين ونشر المعلومات المطلوبة لهذا الحساب. حساب المسار في الزمن الحقيقي هو مفيد في كثير من الحالات. وتشمل هذه العملية التوجيه الذي يحاول الوصول إلى وجهتها، والتقليل من آثار الاصطدام مع العقبات. الأعمال السابقة الخاصة بدراسة أقصر الطرق تقتصر على خوارزميات متسلسلة وموازية على أعمال خدماتية عامة ومرنة. الباحثون اهتموا بطريقة متزايدة في إيجاد حلول عبر الأجهزة. في هذا العمل، نقترح نهجا لتنفيذ خوارزمية ديكسترا التوجيه التي هي فعالة، وذلك باستخدام بطاقة FPGA نوع Virtex 7 ، لتسريع عملية التوجيه، استنادا إلى سرعة الأجهزة (FPGA). نتائج تنفيذ الخوارزمية فيFPGA , (XC7V2000T) 7-Vertix كانت واعدة.

الكلمات المفتاحية :خوارزمية ديكسترا، بطاقة FPGA، اللغة VHDL، WLAN.

Résumé

Avec le développement d'Internet, on assiste aujourd'hui à une véritable explosion du trafic sur les réseaux. Cette explosion du trafic n'est pas sans conséquences. Deux problèmes sont à résoudre, le premier problème est lié à la capacité des liens véhiculant l'information, et l'autre celui du routage pour chercher les chemins les plus courts. Ils sont partiellement résolus aujourd'hui par les algorithmes à vecteur de distance et à information d'état de lien. Mais les protocoles de routage actuels prennent peu en compte les qualités de service, et peuvent difficilement s'adapter à un environnement dynamique, qui se caractérise par des pics de charge brefs, intenses. L'objectif du routage est de déterminer une route, soit un ensemble de liens à parcourir, respectant certaines contraintes, pour établir une connexion d'un nœud source vers un nœud destinataire. Le but d'un algorithme de routage est de permettre le calcul de route entre ces deux nœuds au sens d'un certain critère, et la diffusion des informations nécessaires à ce calcul. Le calcul de la trajectoire en temps réel est utile dans plusieurs situations. Il s'agit notamment du processus de routage qui tente d'atteindre sa destination, et de minimiser les effets de collision avec des obstacles. Les travaux antérieurs sur le plus court chemin sont limités à des algorithmes séquentiels et parallèles sur les architectures à usage varié et flexible. Les chercheurs sont de plus en plus intéressés par les solutions de type matériel, dans ce travail, nous proposons une approche pour mettre en œuvre un algorithme de routage qui est celui de Dijkstra, utilisant une carte de développement FPGA de type Virtex pour accélérer le processus de routage en se basant sur la vitesse du matériel (FPGA). Les résultats de l'implémentation de l'algorithme dans une carte FPGA, Xilinx Vertix-7 (XC7V2000T) sont prometteurs.

Mot clés : WLAN, OSPF, Algorithme de Dijkstra, Langage VHDL, FPGA.

Abstract

With the development of the Internet, today we are witnessing to an explosion of traffic over networks. This explosion of traffic is not without consequences. Two problems are to solve the first one is the ability of links carrying information, and the second concern the routing to find the shortest paths. Today, these problems are partially solved by new algorithms (Distance vector and link state).However, current routing protocols take little account of service qualities, and cannot easily adapt to a dynamic environment which is characterised by peaks and intense brief loads. The main goal of routing is to determine a route or a set of links to browse, within certain constraints, to establish a connection with a source node to a destination node. The purpose of a routing algorithm is to allow the calculation of road between the two nodes in the sense of a certain criterion, and dissemination of required information for this calculation. The calculation of real-time trajectory is useful in many situations, which include the routing process that tries to reach its destination and to minimise the collision effects with obstacles.

Several previous studies on the shortest path are limited to sequential and parallel algorithms on varied and flexible architectures. Currently, researchers are interested in the hardware-based solutions, in this study; we propose an approach to implementing a routing algorithm which is that of Dijkstra, using an FPGA development board type Virtex, to accelerate routing process, based on the speed of the hardware (FPGA).The results of the algorithm implementation in an FPGA Xilinx Vertix-7 (XC7V2000T) are very promising.

Keywords: WLAN, OSPF, Dijkstra algorithm, Language VHDL, FPGA.

Liste des figures

Chapitre 1

Figure 1.1 : couche réseau du modèle OSI 4
Figure 1.2 : encapsulation 4
Figure 1.3: désencapsulation 5
Figure 1.4 : Processus de routage 6
Figure 1.5 : Type de routage 6
Figure 1.6 : Fonctionnalités du protocole OSPF 9
Figure 1.7 : les routeurs échangent des paquets Hello 15
Figure 1.8 : les routeurs échangent des LSA 9
Figure 1.9 : Créer la base de données topologique 15
Figure 1.10 : exécuter de l'algorithme SPF et créer l'arborescence SPF 6
Figure 1.11 : Encapsulation des messages OSPF 9
Figure 1.13 création de contiguïtés avec chaque voisin 15
Figure 1.14 : protocole OSPF à zone unique 9
Figure 1.15 : protocole OSPF à zone multiple 15
Figure 1.16 : la modification du lien affecte uniquement la zone locale 15

Chapitre 2

Figure 2.1 Connectivité d'un noud.....22
Figure 2.2 Projection du flux sur le graphe 4
Figure 2.3 Routage mono chemin et multi chemin 4
Figure 2.4 : Algorithme de Dijkstra généralisé 5
Figure 2.5 : graphe1.....23
Figure 2.6: graphe2.....24
Figure 2.7 : graphe3.....24
Figure 2.8 : graphe4.....25
Figure 2.9 : graphe5.....26
Figure 2.10: graphe6.....26
Figure 2.11 : graphe7.....27
Figure 2.12 : graphe8.....28
Figure 2.13: graphe9.....28
Figure 2.14 : graphe10.....29
Figure 2.15 : graphe11.....29
Figure 2.16 : graphe12.....30

Chapitre 3

Figure 3. 1 : Diagramme d'évolution des coûts 33

Figure 3. 2 : Les différents types de mémoires 34
Figure 3.3 : Schéma comparatif d'un DSP et d'un FPGA 35
Figure 3. 4 : Diagramme des différents types de circuits logiques programmables 35
Figure 3. 5 : Critères de choix du circuit logique programmable FPGA 36
Figure 3. 6 : Différent domaines d'application des FPGA 37
Figure 3.7 : Statistiques du marché occupé par les vendeurs d'FPGA 37
Figure 3.8 : Reprogrammabilité sur site d'un FPGA 38
Figure 3.9 : Caractéristiques des technologies SRAM 39
Figure 3.10 : Caractéristiques des technologies FLASH.....40
Figure 3.11 : Caractéristiques des technologies ANTI-FUSIBLES 41
Figure 3.12 : Architecture interne d'un FPGA 42
Figure 3.13 : Différentes architectures des FPGA.....43
Figure 3.14 : Exemples d'association entre FPGA 44
Figure 3.15 : Exemples d'association entre FPGA et Microprocesseur. 45
Figure 3.16 : Exemple d'implémentation sur LUT 47
Figure 3.17 : Exemple d'implémentation sur des multiplexeurs. 48
Figure 3.18 : Architecture d'un CLB de familles Virtex 48
Figure 3.19: Architecture d'un SLICE de familles Virtex 49
Figure 3.20 : Architecture d'un IOB de familles Virtex 50
Figure 3.21 : Exemple de référence de la famille VIRTEX 51
Figure 3.22 : L'architecture utilisé dans le Virtex 7-2000T 51
Figure 3.23 : Une vue de Virtex-7 2000T de XILINX 52
Figure 3.24 : Une vue de dessus et en bas de Virtex-7 dispositif de 2000T de XILINX,. 52
Figure 3.25 : Comparaison entre une seule Virtex-7 2000T et 4 CIs monolithiques 53
Figure 3.26 : Carte de développement Virtex-7 XC7V2000T 53

Chapitre 4

Figure 4.1 : Dessin au micron d'un inverseur CMOS 78
Figure 4.2 : Exemple de description textuelle 79
Figure 4.3 : Description schématique du même circuit 79
Figure 4.4 : Description fonctionnelle du même circuit.....83
Figure 4.5 : Exemple VHDL/ VERILOG.....78
Figure 4.6 : structure de base d'un module sous VHDL 79
Figure 4.7 : Syntaxe déclarative de l'entité. 79
Figure 4.8 : Syntaxe déclarative de l'architecture. 83
Figure 4.9 : Structure d'un programme sous VHDL 78
Figure 4.10 : Mode d'exécution matériel des outils de CAO 79
Figure 1.11: Méthodologie de conception hiérarchique Top-Down et Bottom-Up 79

Figure 4.12: Etapes de conception sur FPGA 83

Chapitre 5

Figure 5.1 : Les étapes d'implémentation d'un circuit sur FPGA sous l'outil ISE 85
Figure 5.2 : L'étape modifiée de l'Algorithme de DIJKSTRA classique 87
Figure 5.3 : Modélisation du problème 88
Figure 5.4 : Les quatre étapes mentionnées 89
Figure 5.5: Schéma de l'algorithme de dijkstra.....88
Figure5.6 : Exemple de l'application de Dijkstra 89
Figure 5.7 : exemple de l'application sur les réseaux importants .....
Figure 5. 8 : vue externe(RTL) du bloc ROM 89
Figure 5. 9 : vue interne (RTL) du bloc ROM 89
Figure 5.10 : résultat de simulation de bloc ROM 89
Figure 5.11 : vue externe (RTL) du bloc RAM1.....90
Figure 5. 12 : vue interne (RTL) du bloc RAM1 90
Figure 5. 13 : résultat de simulation de bloc mémoire 90
Figure 5. 14 : vue externe (RTL) du bloc opérateur.....91
Figure 5. 15 : vue interne (RTL) du bloc Opérateur.....91
Figure 5. 16 : résultat de simulation de bloc sum_dis 91
Figure 5. 17 : vue externe (RTL) du bloc Comparateur. 91
Figure 5. 18 : vue interne (RTL) du bloc comparateur 92
Figure 5. 19 : résultat de la simulation de bloc comparateur 92
Figure 5. 20 : vue externe du circuit 92
Figure 5.21 : vue interne schéma RTL de l'implémentation de l'algorithme sur Virtex 7 93
Figure 5. 22: Résultats de simulation de l'implémentation de l'algorithme sur FPGA 93
Figure5. 23 : Temps d'exécutions de l'algorithme de DIJKSTRA 92
Figure 5. 24 : Rapport de temps µP /FPGA 93
Figure 5. 25 : configuration du FPGA par un câble JTAG 96

Liste des tableaux

Tableau 1.1 : Table de routage.....11
Tableau 1.2 : Une décomposition du calcul de coût 13
Tableau 1.3 : Le contenu de l'arborescence.....11
Tableau 1.4 : Description de paquet OSPF 13
Tableau 1.5 : plus de routeurs = plus de contiguïtés.....13
Tableau 3. 1 : les facteurs d'évolution des circuits numériques. 37
Tableau 3. 2 : Comparaison des différentes solutions numériques 44
Tableau 3.3 : Avantages et inconvénients des FPGAs 46
Tableau 3.4 : Evolution des FPGAs XILINX famille Virtex 44
Tableau 3.5 : capacité estimé des différents types FPGA 46
Tableau 4. 1 : les valeurs admises par le type standard logic_vector 81
Tableau 5. 1 : l'état du projet de Dijkstra 94
Tableau 5. 2 : ressources internes utilisées par l'implémentation. 94
Tableau 5. 3 : Consommation d'énergie 94

Table Des Matières

Remerciement……I
Dédicace……...II
الملخص…… ……...…III
Résumé…… ……..…IV
Abstract ……...……... V
Liste des figures …… ……..…X
Liste des tableaux IX
Table des matières …… ……...X
Introduction générale……...…....……1

Chapitre 1 : Le routage dans les réseaux de télécommunications

1.1. Introduction……..4
1.2. Introduction aux réseaux de télécommunication..... 4
1.3. Paramètres de la Qualité de Service 6
1.3.1. La bande passante..... 6
1.3.2 Le délai de bout en bout 6
1.3.3 La gigue 6
1.3.4 La perte de données 7
1.4 Intégration de la QoS dans les réseaux 7
1.4.1 Le contrôle de congestion 7
1.4.2 Définition de la congestion 7
1.5 Routage dans les réseaux de télécommunication 8
1.5.1 Principe 8
1.5.2 La détermination du chemin 9
1.5.3 La table de routage 10
1.5.4 Algorithme et métriques de routage 12
1.6. Le protocole OSPF 15
1.6.1 Caractéristiques du protocole OSPF 16
1.6.2 Fonctionnement général d'OSPF 18
1.6.3 Encapsulation des messages OSPF 20
1.6.4 Types de paquets OSPF 21
1.6.5 Paquet Hello 22
1.6.6 Les zones OSPF (unique et zone multiple) 25

1.7 Protocoles dérivés de protocole OSPF (QOSPF) 27
1.8. Conclusion 28

Chapitre 2 : L'algorithme de Dijkstra

2.1. Introduction 29
2.2. Théorie des graphes 29
2.2.1. Graphe 29
2.2.2. Arc orienté 29
2.2.3. Graphe orienté et graphe non orienté 30
2.2.4. Architecture et degré d'un graphe 30
2.2.5. Notion de chemin 30
2.2.6. Connectivité 30
2.2.7. Arbre sous-tendant minimum 31
2.2.8. Choix d'un graphe de réseau 31
2.2.9. Projection des flux et routage 31
2.2.10. Routage 32
2.3. L'algorithme de Dijkstra 32
2.3.1. LE PRINCIPE 33
2.3.2. INITIALISATION 33
2.3.3. I^{ÈME} ÉTAPE 34
2.3.4. (N-2) ÈME ÉTAPE 34
2.3.5. EVALUATION 34
2.4. Description de l'algorithme de Dijkstra
2.5. Conclusion 45

Chapitre 3 : Les circuits logiques programmables

3.1. Introduction 46
3.2. L'évolution des coûts des circuits numériques 46
3.3. Les technologies de Mémorisation 47
3.4. Les circuits logiques Programmables 48
3.5. Les différents types des circuits Logiques Programmables 50
3.6. Les FPGAs 51
3.6.1. Critères de choix du circuit programmable FPGA 51
3.6.2. Différent domaines d'application des FPGA 52
3.6.3. Les fabricants de FPGA 53

3.6.4. Configuration et reconfiguration des FPGA 54
3.6.5. Technologies de programmation des FPGA 54
3.6.5.1. Technologie à base de RAM (XILINX et ALTERA) 54
3.6.5.2. Technologie à base d'EEPROM ou FLASH (LATTICE et ACTEL) 55
3.6.5.3. Technologie à base d'anti-fusibles (ACTEL) 55
3.7. Architecture interne des FPGA 56
3.7.1. Architecture MULTI-COMPOSANTS 58
3.7.2. Avantages et inconvénients des FPGA 59
3.7.3. Les deux grandes familles architecturales d'FPGA 59
3.8. Technologie du premier fondeur d'FPGA « XILINX » 60
3.8.1. Structure des CLB 61
3.8.2. Structure des IOB 61
3.8.3. Nomenclature des Virtex 61
3.8.4. Evolution des FPGAs XILINX famille Virtex 62
3.9. Etude de la VIRTEX 7 62
3.9.1. Présentation de la carte Virtex XC7V2000T 64
3.10. L'apport des FPGA à la télécommunication 65
3.11. Conclusion 66

Chapitre 4 : Les langages de description matérielle et méthodologie d'implémentation

4.1. Introduction 68
4.2. Les étapes d'évolution de La description matérielle 68
4.3. Utilité des HDL 72
4.3.1. Simulation 72
4.3.2. Synthèse 72
4.4. Exemples de langages de description matérielle HDL 73
4.4.1. VHDL (Very High Speed Integrated Circuits Hardware Description Language) 73
4.4.2. Verilog 74
4.4.3. Comparaison entre VHDL et VERILOG et choix 74
4.5. Le langage de description matériel VHDL 75
4.5.1. Structure d'un programme VHDL le couple ENTITY, ARCHITECTURE 77
4.5.2. Éléments du langage VHDL 78
4.5.3. Structure d'un programme sous VHDL 80
4.6. Les TESTBENCHS 81

4.7. Méthodologie de conception 81
4.7.1. Les outils de CAO pour la configuration d'un FPGA 81
4.8. Etapes de conception sur FPGA 82
4.8.1. Spécification du design 83
4.8.2. Développement du design 83
4.8.3. Synthèse 83
4.8.4. Placement et routage 84
4.8.5. Intégration et implémentation 84
4.9. Conclusion 85

Chapitre 5 : Résultats et discussions

5.1. Introduction 86
5.2. Description en VHDL de l'architecture proposée 86
5.3. Présentation de l'environnement de développement 88
5.4. Présentation de la démarche suivie 89
5.4.1. Modélisation de la topologie réseau 90
5.4.2. Les étapes de notre démarche 92
5.4.3. L'architecture de l'algorithme de Dijkstra proposée 93
5.4.4. Fonctionnement de l'architecture 94
5.5. Résultats de l'implémentation de notre architecture sur FPGA 97
5.5.1. Description des blocs et implémentation sur FPGA 97
5.5.1.1. Bloc mémoire 97
5.5.1.2. Bloc opérateur (sum_dis) 99
5.5.1.3. Bloc comparateur (find_min) 100
5.5.1.4. Synthèse du bloc général 101
5.5.2. Le rapport de consommation des ressources 103
5.6. Résultats de comparaisons entre un FPGA et un processeur ordinaire 104
5.7. Transfert de la solution vers un support physique FPGA 106
5.8. Conclusion 106
Conclusion générale et perspectives 108
Liste des publications 111
Références 112

Introduction générale

Introduction générale

Depuis la fin du siècle précédent, La télécommunication couvre la majorité des applications requérant une communication longue distance accessible à tous par une technologique extraordinaire, surtout dans le domaine des transports pour la communication entre des personnes à longue distance, aussi bien en la ville que sur les champs.

De nos jours, les systèmes de télécommunication occupent une place prépondérante dans notre vie, qu'elle soit professionnelle ou privée. On trouve désormais des applications comme; messagerie électronique, transfert de fichiers, des applications permettant de faire transiter du son et l'image. Aujourd'hui, les systèmes de télécommunication sont omniprésents, et il est difficile de trouver un domaine où la télécommunication n'a pas fait son emprunte.

Ces systèmes sont organisés en réseaux, qu'on peut définir comme des ensembles d'équipements de télécommunication et de supports de transmission dont une des fonctions est de permettre le transfert et l'acheminement des différentes informations.

Nous sommes entrés dans l'ère de la télécommunication où le volume et la diversité de ces informations augmentent en exponentielle, et surtout avec la popularité de l'Internet et l'utilisation des nouvelles générations de réseau dans le cadre d'applications en multimédia où le service qualité est garanti, en effet, le routage de l'information est assuré avec une Qualité de Service (QoS) maîtrisée. On dispose d'une série de paramètres, à savoir la bande passante, le délai de bout en bout, la gigue ou le taux de perte des données qui permettent de caractériser les connexions transitant par le réseau, donc, d'autres facteurs agissent dans la transmission de l'information [1]. Ce qui provoque aux routeurs actuels exécutant les algorithmes de routage des difficultés croissantes avec les qualités et les exigences des services liés aux réseaux. Du fait de l'accroissement des exigences réseau, la qualité de service et la complexité du calcul, le mécanisme demande toujours de plus en plus de ressources opératoires. Ces réseaux doivent allier la vitesse du hardware à la flexibilité du software.

Comme nous l'avons vu dans la littérature scientifique, les solutions proposées pour résoudre les problèmes de routage des informations dans les réseaux de télécommunication se basent sur la conception de nouveaux protocoles et algorithmes de routage qui sont des solutions softwares complexes [2][3]. Les chercheurs à conception hard sont convaincus que les problèmes sont au niveau du processeur qui effectue les calculs et non pas au niveau des différents algorithmes.

Vu le développement de la technologie des circuits logiques programmables, les chercheurs s'intéressent aux solutions Hardware, de ce fait, ils proposent d'utiliser la technologie FPGA qui est plus performante que les processeurs classiques, la plupart de ces chercheurs utilisent des cartes FPGAs en parallèle avec un processeur ordinaire pour accomplir certaines tâches spécifiques afin de garder la rapidité requise et conforme de la machine quel que soit la tâche qui lui est demandée [4] [5].

Les protocoles de routage spécifient la manière aux routeurs de communiquer ensemble pour diffuser des informations qui leur permettent de choisir des routes entre deux nœuds. En effet, parmi les algorithmes de recherche du plus court chemin, les plus répandus dans les réseaux sans fil, on a l'algorithme de Dijkstra qui est utilisé par le protocole de routage OSPF [6].

Dans notre travail de thèse, notre objectif est de proposer une nouvelle technique pour l'accélération de processus de routage dans les réseaux de télécommunication en se basant sur la rapidité de matériel (FPGA), en effet, nous avons fait une étude complète avec conception, simulation d'une méthodologie d'implantation de l'un des algorithmes de recherche du plus court chemin, celui de Dijkstra sur un circuit FPGA.

La thèse est répartie en cinq chapitres et sont représentés comme suit :

Le premier chapitre est consacré à l'état de l'art du routage dans les réseaux informatiques, en mettant en évidence les techniques algorithmiques permettant de transférer les données d'un nœud à un autre, ainsi que de maintenir à jour, en fonction de l'évolution réelle de la topologie du réseau, les informations permettant de réaliser ce transfert de manière optimale.

Le deuxième chapitre a pour objectif de définir certains concepts de la théorie des graphes qui servent de base à l'analyse et à la conception topologique des réseaux et de présenter l'algorithme de Dijkstra avec un exemple suffisamment détaillé pour que lecteur puisse comprendre son fonctionnement.

Le troisième chapitre concerne les circuits reconfigurables avec l'architecture des FPGAs avec suffisamment de détails afin de comprendre la complexité du sujet. Ensuite, les exigences du critère de choix entre les différents circuits reconfigurables tels que les FPGAs et les ASICs avec une comparaison des FPGAs et des circuits séquentiels, par suite, on a illustré la contribution des FPGAs pour les réseaux de télécommunications.

Le quatrième chapitre est consacré aux langages de description matérielle et leurs évolutions, les logiciels de conception assistée par ordinateur (CAO) permettent de vérifier le fonctionnement d'un circuit électronique sans l'avoir jamais réalisé matériellement, et les étapes de conception sur FPGA.

Dans le cinquième et dernier chapitre, nous avons détaillé notre approche, à savoir, la méthodologie d'implémentation de l'algorithme de Dijkstra dans un circuit FPGA à l'aide du langage VHDL. Après simulation, nous présenterons les résultats et nous effectuerons une comparaison entre l'implémentation sur FPGA et celle sur un processeur ordinaire.
Ce travail peut être approfondi encore plus loin dans l'amélioration et l'optimisation pour la gestion d'un réseau de nœuds en intégrant un outil embarqué qui permettrait de traiter les données et selon le programme embarqué il ne retiendra que les informations utiles à transmettre au calculateur central, ce qui donne le meilleur compromis entre le nombre de cycle d'exécution en parallèle (vitesse) et le nombre d'opérateurs à générer (surface). Ceci commencera par une définition de métriques adéquates et d'exploiter au maximum les ressources matérielles disponibles pour augmenter les performances en temps réel sous la contrainte de consommation minimale. Une réécriture du code VHDL de certains opérateurs de calcul serait profitable pour accélérer la vitesse du fonctionnement du modèle et minimiser encore les ressources matérielles nécessaires. Il serait intéressant aussi d'adapter et d'optimiser le code VHDL en tenant compte des spécificités des cibles FPGAs utilisées.
Enfin nous clôturons ce document avec une conclusion générale ainsi que des perspectives d'application et d'exploitation de la technologie programmable FPGA qui est prometteuse et pleine de réussite.

Chapitre 1

LE ROUTAGE DANS LES RESEAUX DE TELECOMMUNICATIONS

1.1. Introduction :

L'objectif de ce chapitre est de présenter la problématique générale de routage dans les réseaux de télécommunication. Après une brève introduction aux réseaux de télécommunication, nous décrivons les paramètres et les métriques caractérisant le processus de routage, nous présentons la fonction de routage dans les réseaux de télécommunication ainsi que les algorithmes généralement utilisés. Nous présentons et analysons ensuite le protocole choisi dans notre travail qui est le protocole **O**pen **S**hortest **P**ath **F**irst (OSPF).

1.2. Introduction aux réseaux de télécommunication

Au début des années 70, chaque constructeur a développé sa propre solution réseau autour d'architectures et de protocoles privés (SNA d'IBM [7], DEC net de DEC [7], DSA de Bull, TCP/IP du DoD, …) et il s'est vite avéré qu'il serait impossible d'interconnecter ces différents réseaux si une norme internationale n'était pas établie. Cette norme a été introduite par l'ISO (*International Standard Organization*) [8], il s'agit du modèle OSI [9] [10], *(Open System Interconnection*). Ce dernier décrit la manière dont, matériels et logiciels coopèrent selon une architecture en couches permettant la communication. Ce modèle constitue également une aide pour le dépannage, car il fournit un cadre de référence qui décrit la façon dont les composants sont censés fonctionner. Le modèle OSI comporte sept couches (Figure 1.1).

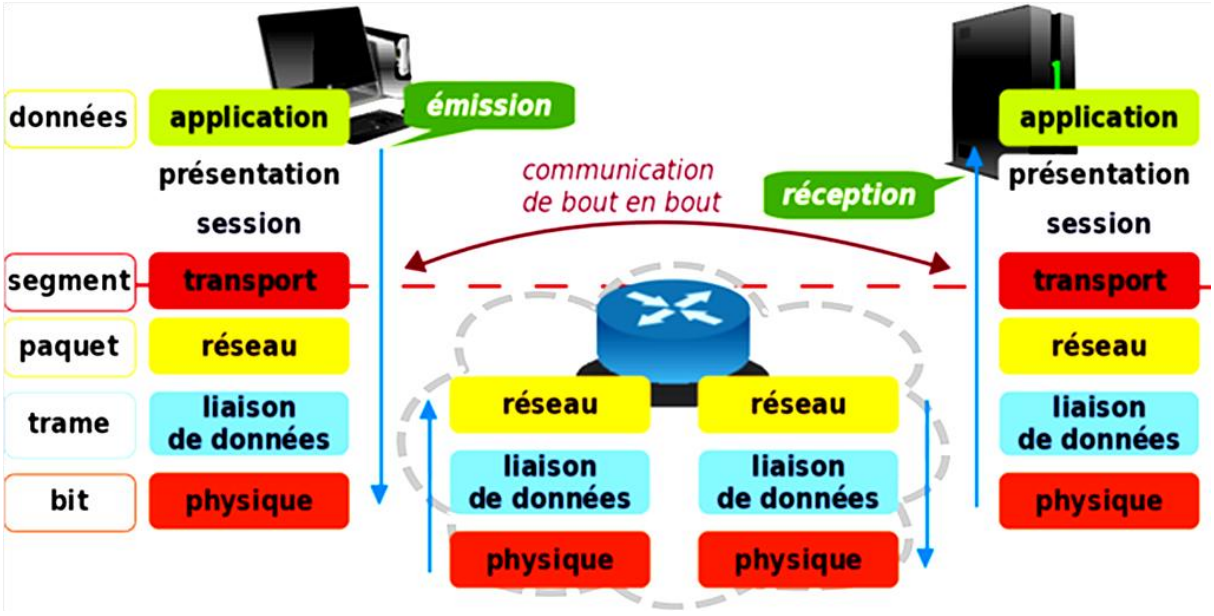


Figure 1.1 : Les 7 couches du modèle OSI [11]

Il s'étend des niveaux les plus bas des techniques de transmission jusqu'aux interactions de haut niveau entre applications spécifiques. Ces sept couches sont :

- **La couche physique** : Elle établit la connexion physique entre un système et le réseau, et concerne la manière dont le message est transporté en fonction du support (câbles, ondes, etc.). A ce niveau, l'unité d'information est le **Bit**.
- **La couche de liaison de données** : Cette couche "fait" et "défait" les paquets de données à transmettre. Elle assure le contrôle et la correction des données, le contrôle des accès vers la carte réseau. Sur cette couche, le message est appelé une **trame**.
- **La couche réseau** : Elle assure le routage des paquets et gère la relation entre les adresses logiques utilisées par l'utilisateur et les adresses physiques implantées dans les machines. Elle se charge aussi des problèmes issus des éventuelles incompatibilités entre les couches basses des réseaux traversés (taille de trame, adressage physique différent, etc.).
- **La couche transport** : Elle reçoit les données de la couche session et assure leur transport de bout en bout. Elle peut éventuellement fragmenter les données en plusieurs paquets afin de faciliter leur transmission et s'assure de leurs arrivées à destination. Pour améliorer la rapidité de la transmission, la couche transport peut créer plusieurs connexions simultanées sur lesquelles elle répartit les paquets (principe de multiplexage). Elle assure la transparence des réseaux traversés vis à vis des couches hautes.
- **La couche session** : Il s'agit de la première couche qui s'intéresse à la communication proprement dite. Elle ouvre une session avec la station de destination. Cette session reste ouverte durant toute la communication. Lorsque la station de destination envoie un accusé de réception global, il y a fermeture de la session. La couche offre des services de synchronisation et de rattrapage d'erreurs.
- **Couche présentation** : Celle-ci traite de la mise en forme de l'information. Au-dessus d'elle, l'information est codée sous une forme plus appropriée pour son transport. Ainsi, la compression des données et le cryptage sont du ressort de la couche 6.
- **La couche application** : Elle est chargée d'offrir à l'utilisateur les fonctions de communication. Ces fonctions sont le transfert de fichier, la messagerie, l'émulation de terminal virtuel, l'exécution de travaux à distance, etc. La couche 7 représente l'interface utilisateur pour les fonctions de communication.

Les principes qui ont conduit à ces 7 couches sont les suivants :

- Une couche doit être créée lorsqu'un nouveau niveau d'abstraction est nécessaire,
- Chaque couche a des fonctions bien définies,
- Les fonctions de chaque couche doivent être choisies dans l'objectif de la normalisation internationale des protocoles,

- Les frontières entre couches doivent être choisies de manière à minimiser le flux d'information aux interfaces,
- Le nombre de couches doit être tel qu'il n'y ait pas cohabitation de fonctions très différentes au sein d'une même couche et que l'architecture ne soit pas trop difficile à exploiter.

Les couches basses (1, 2, 3 et 4) sont nécessaires à l'acheminement des informations entre les extrémités concernées et dépendent du support physique. Les couches hautes (5, 6 et 7) sont responsables du traitement de l'information relative à la gestion des échanges entre systèmes informatiques. Par ailleurs, les couches 1 à 3 interviennent entre machines voisines, et non entre les machines d'extrémité qui peuvent être séparées par plusieurs routeurs. Les couches 4 à 7 sont au contraire des couches qui interviennent entre hôtes distants.

1.3. Paramètres de la Qualité de Service

La qualité de service QoS (**Q**uality **o**f **S**ervice) comporte une série de paramètres qui permettent de caractériser les garanties qui peuvent être fournies à chaque flux (ou connexion) transitant par un réseau de télécommunication. Ces paramètres sont généralement la bande passante, le délai de bout en bout, la gigue, la perte de données.

1.3.1. La bande passante

La bande passante est la quantité maximale de données pouvant être transmise d'une source vers une destination en une unité de temps. Il s'agit en fait du minimum des bandes passantes disponibles sur les liens composant le chemin de la source à la destination. La bande passante disponible sur un chemin dépend donc du support des liaisons, mais aussi du nombre et du débit des flux qui partagent ces liaisons. [12]

A titre d'exemple, les applications vidéo requièrent généralement une bande passante élevée car d'une part, une image représente une grande quantité d'informations et d'autre part, le nombre d'images envoyées par unité de temps doit être suffisant pour obtenir une vision satisfaisante.

1.3.2 Le délai de bout en bout

Appelé aussi latence ou temps de réponse, il s'agit de la durée nécessaire à l'acheminement d'un paquet de données de bout en bout. Cette durée dépend de la qualité du support des liaisons, mais aussi du temps passé dans les différentes files d'attente du chemin.

Par conséquent, le délai augmente avec la charge du réseau, par exemple, les applications de conversation sont particulièrement sensibles au délai. On estime qu'une conversation devient pénible si le temps d'acheminement dépasse 28 ms [13].

1.3.3 La gigue

La gigue est la variation du délai de bout en bout, issue des congestions instantanées lorsque plusieurs flux de données sollicitent au même moment le même port de sortie. Un bon contrôle

de la gigue est souhaité dans les applications de type vidéo à la demande, car une forte gigue entraîne des distorsions lors de l'écoute ou de la visualisation des médias.

1.3.4 La perte de données

Le taux de perte est la proportion des paquets qui ne parviennent pas à leur destination. Ces pertes dépendent de :

- La fiabilité du support des liaisons : un support peu fiable entraîne une dégradation fréquente des paquets. Si un paquet s'avère corrompu à l'issue de sa vérification grâce aux bits de contrôle (checksum), il sera détruit. On peut toutefois estimer qu'actuellement la fiabilité des liens dans le réseau Internet est relativement élevée, et que par conséquent, les pertes de paquets dues à des défauts de support sont très faibles [14].
- L'occurrence de surcharges locales (congestions) dans le réseau : les paquets devant être placés dans une file d'attente pleine sont détruits.

Lorsque le protocole de transport garantit l'arrivée de l'information transmise, les paquets perdus doivent être renvoyés. Les pertes ont donc une influence directe sur l'augmentation du délai et sur la gigue.

1.4 Intégration de la QoS dans les réseaux

Le problème de la qualité de service dans les réseaux constitue un vaste sujet de recherche et a fait l'objet de nombreuses techniques proposées dans la littérature. En dehors des techniques tendant à organiser l'architecture de communication de façon à réduire les délais d'acheminement des messages (le processus de routage) tout en garantissant une bonne disponibilité du support de transmission.

1.4.1 Le contrôle de congestion

Dans les réseaux de télécommunication, les protocoles de routage généralement utilisés ne tiennent pas compte de la charge du réseau. Par conséquent, il arrive que le débit des flux partageant une liaison soit trop grand, entraînant une surcharge locale du réseau. Ces surcharges sont appelées congestions.

I.4.2 Définition de la congestion

Lorsqu'un paquet arrive dans un routeur, il est placé dans une file d'attente. Si le débit des paquets en entrée d'un routeur est plus grand que le débit de sortie, le nombre de paquets dans la file d'attente augmente. La taille des files étant limitée, une congestion survient lorsque la file est pleine. Les paquets devant être placés dans la file pleine sont alors détruits.

Il est donc essentiel de limiter d'une part, l'apparition de congestions et d'autre part, leur durée. De nombreux travaux se sont orientés dans ce sens, notamment ceux de Van Jacobson [15] [16].

Ils ont donné naissance à un certain nombre de techniques permettant de moduler le débit des sources, et ainsi limiter l'aspect néfaste des congestions [17].

1.5 Routage dans les réseaux de télécommunication

1.5.1 Principe

Le routage est l'une des fonctionnalités principales de la couche réseau qui a la responsabilité de décider sur quelle ligne de sortie, un paquet entrant doit être retransmis.

D'une manière générale, on distingue la *remise directe*, qui correspond au transfert d'un datagramme entre deux ordinateurs du même réseau, et la *remise indirecte* qui est mise en œuvre quant au moins un routeur sépare l'expéditeur initial et le destinataire final. Pour sa part, la remise indirecte nécessite de déterminer vers quel routeur envoyer un datagramme IP en fonction de sa destination finale. Ceci est rendu possible par l'utilisation d'une *table de routage* spécifique à chaque routeur, permettant de déterminer vers quelle voie de sortie envoyer un datagramme destiné à un réseau quelconque.

Les routeurs permettent d'interconnecter les segments d'un réseau ou des réseaux entiers. Leur rôle consiste à acheminer les trames de données entre les réseaux, en fonction des informations de la couche 3. Ils prennent des décisions logiques quant au meilleur acheminement possible des données, puis redirigent les paquets vers le port de sortie approprié afin qu'ils soient encapsulés pour la transmission. Les phases d'encapsulation et de désencapsulation se produisent à chaque passage d'un paquet dans un routeur. Le routeur doit en effet désencapsuler la trame de données de la couche 2 pour accéder à l'adresse de couche 3 et l'examiner. L'encapsulation consiste à fractionner le flux de données en segments et ajouter les en-têtes et les en-queues appropriés avant de transmettre les données. Le processus de désencapsulation, quant à lui, consiste à retirer les en-têtes et les en-queues, puis à recombiner les données en un flux continu (voir figure1.2 et figure1.3).

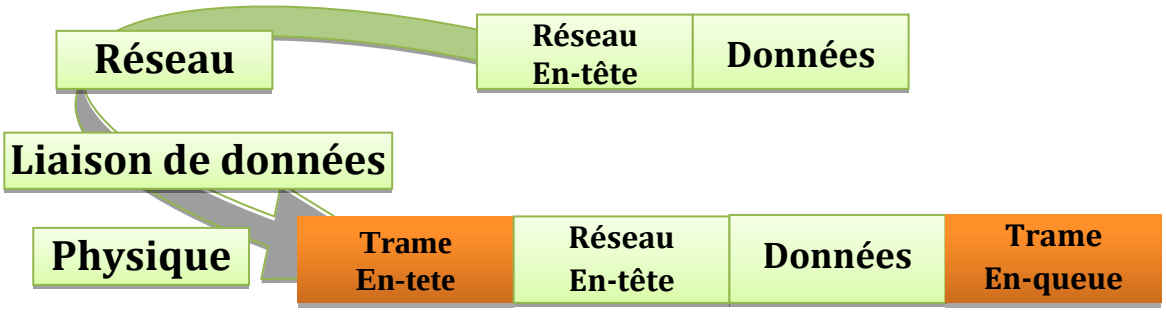


Figure 1.2 : encapsulation

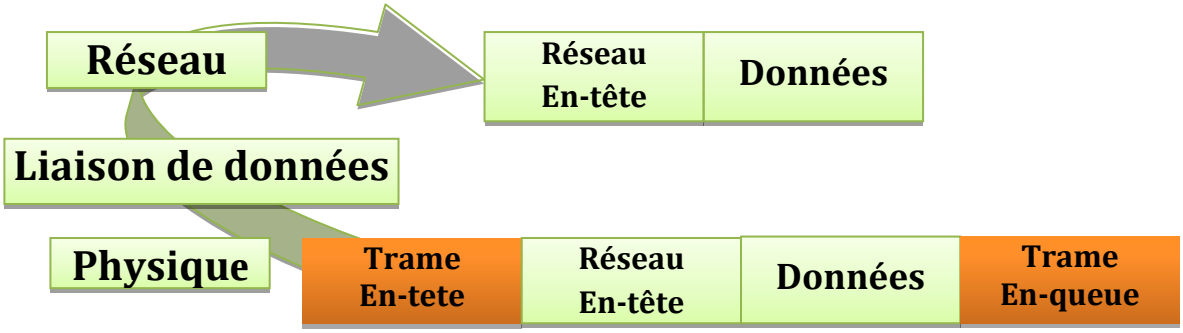


Figure 1.3 : désencapsulation

1.5.2 La détermination du chemin

La détermination du chemin se produit au niveau de la couche réseau. Ce processus permet au routeur de comparer l'adresse de destination aux routes disponibles dans sa table de routage et de choisir le meilleur chemin possible. Les routeurs acquièrent ces chemins soit par l'intermédiaire du routage statique, soit par l'intermédiaire du routage dynamique. Les chemins configurés manuellement par l'administrateur réseau sont appelés « routes statiques ». Ceux que le routeur a acquis d'autres chemins à l'aide d'un protocole de routage sont dits « routes dynamiques ».

La détermination du chemin permet au routeur de choisir le port à partir duquel d'envoyer un paquet pour que celui-ci arrive à destination. On appelle ce processus le routage d'un paquet. Chaque routeur rencontré sur le chemin du paquet est appelé un saut.

Les processus impliqués dans la sélection du chemin pour chaque paquet sont illustrés dans la figure I.4 :

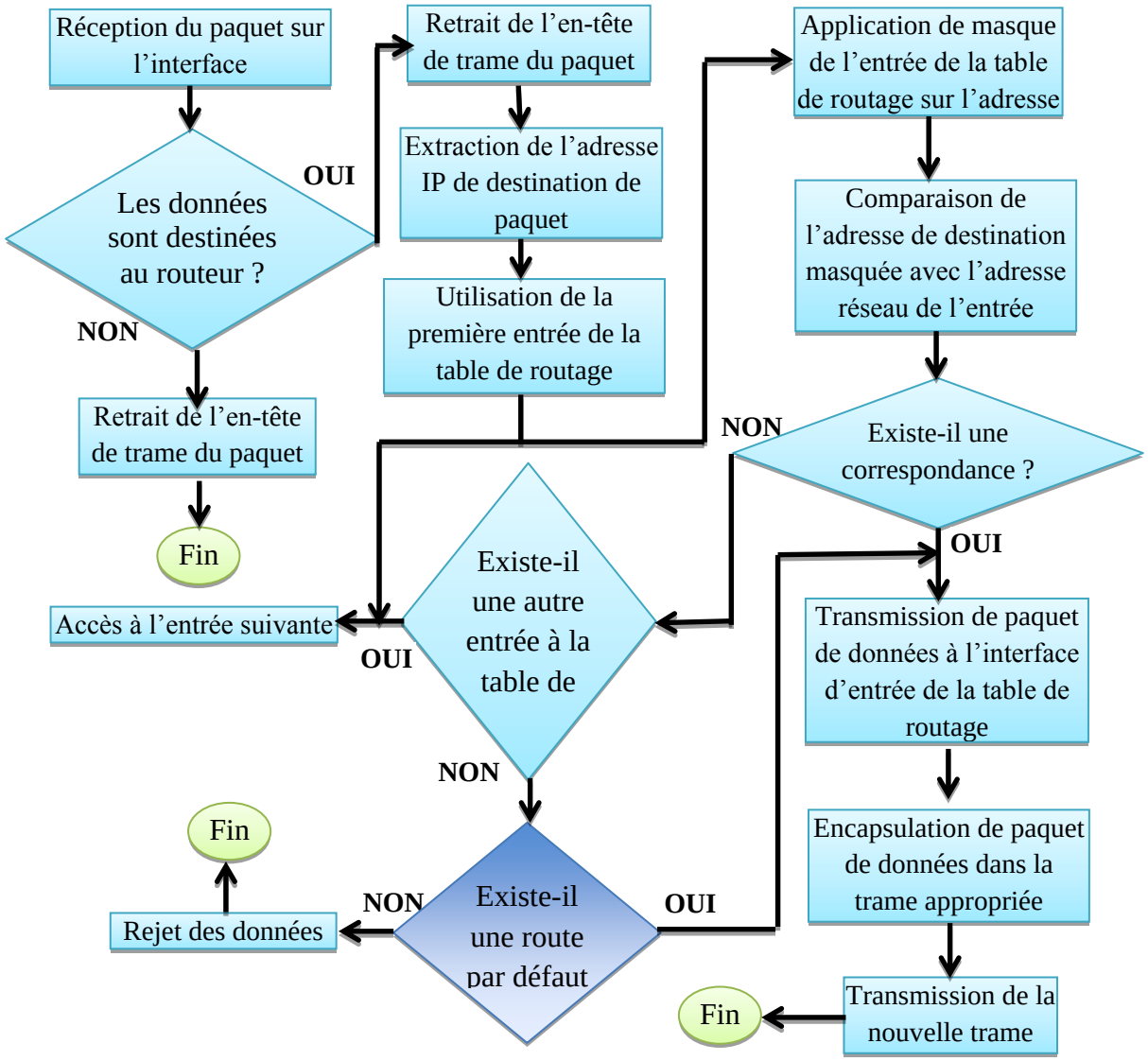


Figure I.4 : Processus de routage

1.5.3 La table de routage

Les routeurs emploient des protocoles pour construire et gérer les tables de routage contenant les informations d'acheminement. Le processus de sélection du chemin en est ainsi facilité. Les protocoles de routage placent diverses informations d'acheminement dans les tables de routage. Le contenu de ces informations varie selon le protocole de routage utilisé.

L'essentiel du contenu d'une table de routage est constitué de quadruplets *(destination, passerelle, masque, interface)* où :

- **Destination :** est l'adresse IP d'une machine ou d'un réseau de destination.
- **Passerelle :** (Gateway) est l'adresse IP du prochain routeur vers lequel on envoie le datagramme pour atteindre cette destination.
- **Masque :** est associé au réseau de destination.

- **Interface :** désigne l'interface physique par laquelle le datagramme doit réellement être y.

Une table de routage contient notamment une *route par défaut* qui spécifie un routeur vers lequel sont envoyés tous les datagrammes pour lesquels il n'existe pas de route dans la table.

Tous les routeurs mentionnés dans une table de routage doivent être directement accessibles à partir du routeur considéré. Cette technique, dans laquelle un routeur ne connaît pas le chemin complet menant à une destination, mais simplement la première étape de ce chemin, est appelée routage par sauts successifs (*next-hop routing*).

Exemple de table de routage (tableau 1.1)

Destination réseau	Masque réseau	Adr. passerelle	Adr. Interface
0.0.0.0	0.0.0.0	192.168.8.254	192.168.8.16
192.168.8.0	127.252.255.0	On-link	192.168.8.16
192.168.8.16	127.252.255.255	255.255.255.255	192.168.8.16

Tableau 1.1: Table de routage

L'ordre de traitement de la table de routage va **des masques les plus longs aux plus petits**. C'est à dire que le routeur va d'abord comparer les sous-réseaux avec le masque 255.255.255.255 pour finir par comparer les sous-réseaux avec le masque 0.0.0.0.

La table de routage simplifiée ci-dessus peut se traduire ainsi (dans l'ordre de traitement) :

1. Route vers l'ordinateur lui-même, "Destination réseau" et "Adresse interface" ont la même valeur. On remarquera le masque entièrement à 255 (/32 en CIDR) qui permet de désigner un réseau (une plage) **limitée à une seule adresse**.
2. Route vers le réseau local sur lequel est connecté l'ordinateur. "On-link" indique que l'ordinateur est directement connecté au réseau concerné, il n'y a donc pas besoin de routeur pour l'atteindre.
3. Route par défaut 0.0.0.0/0.0.0.0 : c'est la route utilisée si aucune autre route possible n'a été trouvée dans la table de routage.

La taille d'une table de routage dépend de la taille du réseau. Par conséquent, afin d'accélérer la recherche de correspondances parmi les entrées d'une table, il est préférable que sa taille soit réduite. A ses débuts, l'Internet regroupait un petit nombre de machines isolées et de petits réseaux [18]. Les tables de routage étaient donc de petites tailles. Un seul protocole assurait le routage, GGP (Gateway To Gateway Protocol). Avec la fantastique expansion qui s'est produite, l'explosion des tables de routage fut l'une des raisons pour lesquelles la décision de découper l'Internet en un ensemble d'entités indépendantes fut prise.

I.5.4 Algorithme et métrique de routage

L'algorithme est le résultat d'une démarche logique de résolution d'un problème pour la mise en œuvre pratique sur ordinateur et afin d'obtenir des résultats concrets, il faut passer par l'intermédiaire d'un langage spécifique de programmation. Les algorithmes de routage utilisés pour définir le port pour envoyer un paquet selon les protocoles de routage. Ils reposent sur l'utilisation de métriques pour prendre ce type de décision.

Les protocoles de routage sont conçus pour répondre à un ou plusieurs des objectifs suivants :

- *Optimisation* : capacité d'un algorithme de routage à sélectionner le meilleur chemin. Ce dernier sera choisi en fonction des métriques et de la pondération utilisées dans le calcul. Par exemple, un algorithme peut utiliser à la fois le nombre de sauts et le délai comme métriques, mais considérer que le délai doit prévaloir dans le calcul.
- *Simplicité et réduction du temps système :* plus l'algorithme est simple et plus il sera traité efficacement par le processeur et la mémoire du routeur. Ce paramètre est important si le réseau veut pouvoir évoluer vers des proportions plus conséquentes, comme l'Internet.
- *Efficacité et stabilité* : un algorithme de routage doit pouvoir fonctionner correctement dans des circonstances inhabituelles ou imprévues, comme les défaillances de matériels, les surcharges et les erreurs de mise en œuvre.
- *Flexibilité* : un algorithme de routage doit pouvoir s'adapter rapidement à toutes sortes de modifications du réseau, touchant par exemple la disponibilité et la mémoire du routeur, la bande passante ou le délai réseau.
- *Rapidité de convergence* : la convergence est le processus par lequel tous les routeurs s'entendent sur les routes disponibles. Lorsqu'un événement sur le réseau entraîne des modifications au niveau de la disponibilité d'un routeur, des mises à jour sont nécessaires afin de rétablir la connectivité du réseau. Une convergence lente des algorithmes de routage peut empêcher la livraison des données.

Chacun d'eux interprète à sa façon ce qui est le mieux. L'algorithme génère un nombre, appelé valeur métrique, pour chaque chemin traversant le réseau. Les algorithmes de routage perfectionnés effectuent la sélection du chemin en fonction de plusieurs métriques combinées en une valeur composite. Généralement, les valeurs métriques indiquent le meilleur chemin.

Les métriques peuvent être calculées sur la base d'une seule caractéristique de chemin, comme elles peuvent l'être sur la base de plusieurs caractéristiques. Les métriques les plus communément utilisées par les protocoles de routage sont les suivantes :

- *Bande passante* : la bande passante représente la capacité de débit d'une liaison. Une liaison Ethernet de 10 Mbits/s est généralement préférable à une ligne louée de 64 Kbits/s.
- *Délai* : le délai est le temps nécessaire à l'acheminement d'un paquet, pour chaque liaison, de la source à la destination. Il dépend de la bande passante des liaisons intermédiaires, de la quantité de données pouvant être temporairement stockées sur chaque routeur, de la congestion du réseau et de la distance physique.
- *Charge* : la charge est la quantité de trafic sur une ressource réseau telle qu'un routeur ou une liaison.
- *Fiabilité* : la fiabilité se rapporte habituellement au taux d'erreurs de chaque liaison du réseau.
- *Nombre de sauts* : le nombre de sauts est le nombre de routeurs par lesquels un paquet doit passer avant d'arriver à destination. Dans la table de routage, parmi les routes empruntées, chaque routeur équivaut à un saut. Un nombre de sauts égal à 4 signifie que les données doivent passer par quatre routeurs pour atteindre leur destination. Lorsque plusieurs chemins sont possibles, c'est le chemin comportant le moins de sauts qui est privilégié.
- *Tops* : délai d'une liaison de données utilisant les tops d'horloge d'un PC IBM, un top d'horloge correspondant environ à 1/18 seconde.
- *Coût* : le coût est une valeur arbitraire, généralement basée sur la bande passante, une dépense monétaire ou une autre mesure, attribuée par un administrateur réseau.

A la réception d'un paquet par un routeur, ce dernier doit déterminer vers quel routeur le transmettre, à partir de la table de routage construite par l'algorithme de routage utilisé. La formulation générale d'une telle procédure s'écrit :

```
Procédure Routage IP (données Dat : datagramme, Tab, Table de routage)
Début
    D : = adresse IP de destination de Dat
    Si D appartient à l'un des réseaux directement accessibles
        Alors // remise directe
        Envoyer Dat vers l'adresse IP, en adresse physique, encapsulation de Dat dans une
        physique et émission de la trame via l'interface physique correspondante
        Sinon // remise indirecte
            Pour chaque entrée de Tab faire
                N : = (D + du masque) résultat 'et logique' de D et du masque de sous-réseau
                Si N = l'adresse réseau de la destination de l'entrée
                    Alors
                    Routeur Dat vers cette destination
                    Sortir
                finsi
            finpour
        Si aucune correspondance n'est trouvée
            Alors
            Retourner à l'application d'origine du datagramme une erreur de routage (machine
            inaccessible, réseau inaccessible, etc.).
        Finsi
    Finsi
Fin
```

Le fait de considérer le réseau Internet comme un système homogène et unifié, a rapidement posé des problèmes. Tout d'abord, il est évident que plus le réseau est grand, c'est-à-dire regroupant plus de nœuds et de destinations potentielles, plus la probabilité d'un changement de topologie du réseau est grande (ajout d'un nœud, d'un lien ou panne). Le routage est donc susceptible de changer plus fréquemment. Le choix et la mise à jour des protocoles deviennent particulièrement complexes, surtout si les organisations propriétaires des portions du réseau ne parviennent pas à s'entendre.

Toutes ces raisons ont conduit au découpage du réseau Internet en un ensemble d'entités indépendantes appelées Systèmes Autonomes (AS) [19], que nous désignerons aussi sous le nom

de domaines, chacun d'entre eux étant sous une administration unique. Les protocoles sont unifiés au sein d'un même AS, ce qui permet d'assurer un routage efficace.

De plus, l'un des AS du réseau Internet permet d'interconnecter tous les autres. Ce système autonome particulier est appelé épine dorsale (backbone). Ainsi, tous les AS du réseau Internet sont interconnectés (Figure I.5). On nomme les nœuds (ou routeurs) appartenant à un AS des passerelles (ou gateways) internes. Les routeurs permettant l'interconnexion entre deux AS sont appelés passerelles (ou gateways) externes.

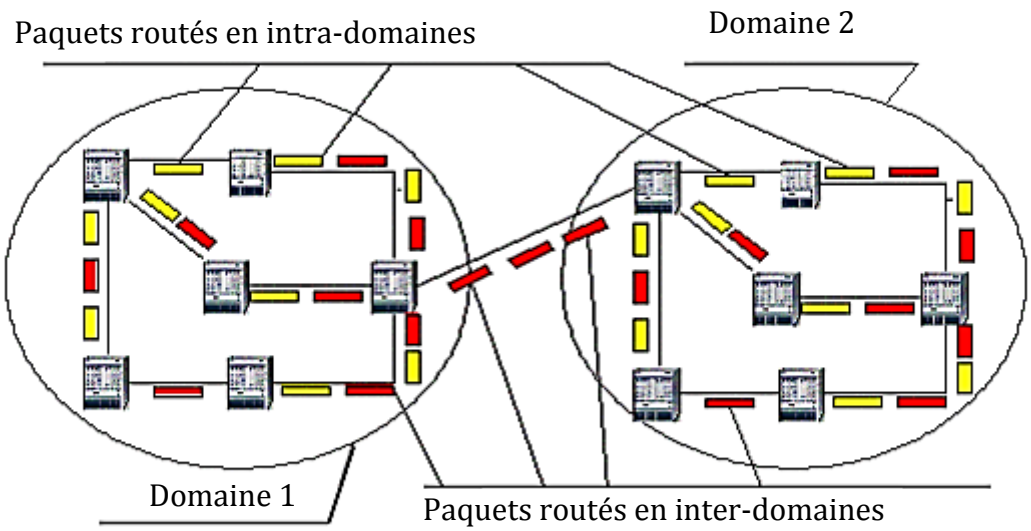


Figure I.5 : Type de routage

Le calcul des tables de routage est effectué indépendamment grâce à un protocole de routage. Ce calcul permet de déterminer le plus court chemin entre deux nœuds du réseau. Il existe de nombreux protocoles de routage, que l'on peut diviser en deux catégories :

- Les protocoles intra-domaines "interior Gateway Protocol (IGP)" (tels que RIP, OSPF [20], IGRP, IS-IS [21]) qui calculent les tables de routage au sein d'un même AS.
- Les protocoles inter-domaines "Exterior Gateway Protocol (EGP)" (tels que EGP [22] ou BGP [23]) calculent les tables de routage entre des nœuds appartenant à des AS différents.

Dans ce mémoire, nous nous focaliserons plus particulièrement sur les protocoles intra-domaines, que l'on peut répartir entre deux grandes familles :

✓ Les protocoles à vecteur de distance.

✓ Les protocoles à états de liens.

1.6 Le protocole OSPF

OSPF (Open Shortest Path First) est un protocole de routage dynamique défini par l'IETF (Internet Engineering Task Force) à la fin des années 80.

Le protocole OSPF est un protocole de routage à état de liens qui a été développé pour remplacer le protocole de routage à vecteur de distance RIP. Le protocole RIP était un protocole de routage acceptable au début des réseaux et d'Internet. Cependant, le fait que le protocole RIP se basait uniquement sur le nombre de sauts comme seule métrique pour déterminer la meilleure route est rapidement devenu problématique. L'utilisation du nombre de sauts n'est pas adaptée aux réseaux de grande taille avec plusieurs chemins de vitesses variables. Le protocole OSPF présente des avantages considérables par rapport au protocole RIP car il offre une convergence plus rapide et s'adapte mieux aux réseaux de plus grande taille [24].

Ce protocole a deux caractéristiques essentielles :

- Il est ouvert : c'est le sens du terme Open d'OSPF. Son fonctionnement est connu de tous.
- Il utilise l'algorithme SPF pour Shortest Path First, plus connu sous le nom d'algorithme de Dijkstra, afin d'élire la meilleure route vers une destination donnée.

Plus Courts Chemins et Métrique OSPF

Le protocole OSPF utilise le coût comme métrique. Un coût plus faible indique un meilleur chemin qu'un coût plus élevé.

Le coût d'une interface est inversement proportionnel à la bande passante de l'interface. Par conséquent, une bande passante plus élevée indique un coût plus faible. Une surcharge et des délais supplémentaires correspondent à un coût supérieur. Ainsi, une ligne Ethernet 10 Mbit/s présente un coût plus élevé qu'une ligne Ethernet 100 Mbit/s.

La formule utilisée pour calculer le coût OSPF est la suivante :

- **Coût** = bande passante de référence / bande passante de l'interface

La bande passante de référence par défaut correspond à 10^8 (100 000 000) ; par conséquent, la formule est la suivante :

$$\delta(interface) = \frac{10^8}{bande\ passante\ de\ l'interface_{bits/s}}$$

La métrique peut aussi être définie manuellement.

Le tableau 1.2 nous montre comment définir manuellement la métrique. On remarque que les interfaces Fast Ethernet, Gigabit Ethernet et 10 GigE partagent le même coût, étant donné que la valeur de coût OSPF doit être un entier. Par conséquent, étant donné que la bande passante de référence par défaut est définie sur 100 Mbit/s, tous les liens qui sont plus rapides que Fast Ethernet ont également un coût de 1.

Type d'interface	Bande passante de référence en bits/s		Bande passante par défaut en bits/s	Coût
10 Gigabit Ethernet **10 Gbit/s**	100 000 000	÷	100 000 000	1
Gigabit Ethernet **1 Gbit/s**	100 000 000	÷	100 000 000	1
Fast Ethernet **100 Mbit/s**	100 000 000	÷	100 000 000	1
Ethernet **10 Mbit/s**	100 000 000	÷	10 000 000	10
Série **1 544 Mbit/s**	100 000 000	÷	1 544 000	64
Série **128 Kbit/s**	100 000 000	÷	128 000	781

Coût identique grâce à la bande passante de référence

Tableau 1.2 : Une décomposition du calcul de coût

1.6.1. Caractéristiques du protocole OSPF

Les caractéristiques du protocole OSPF, comme illustré dans la Figure 1.6 :

- *Sans classe* : Il est sans classe par conception ; par conséquent, il prend en charge VLSM et CIDR.
- *Efficace* : Les changements de routage déclenchent des mises à jour de routage (pas de mises à jour régulières). Il utilise l'algorithme SPF pour déterminer le meilleur chemin.
- *Convergence rapide* : Il diffuse rapidement les modifications apportées au réseau.
- *Évolutif* : Il fonctionne bien sur les petits et grands réseaux. Les routeurs peuvent être regroupés en zones pour prendre en charge un système hiérarchique.
- Sécurisé : Il prend en charge l'authentification MD5 (Message Digest 5). Une fois activés, les routeurs OSPF acceptent uniquement les mises à jour de routage chiffrées des homologues avec le même mot de passe pré-partagé.

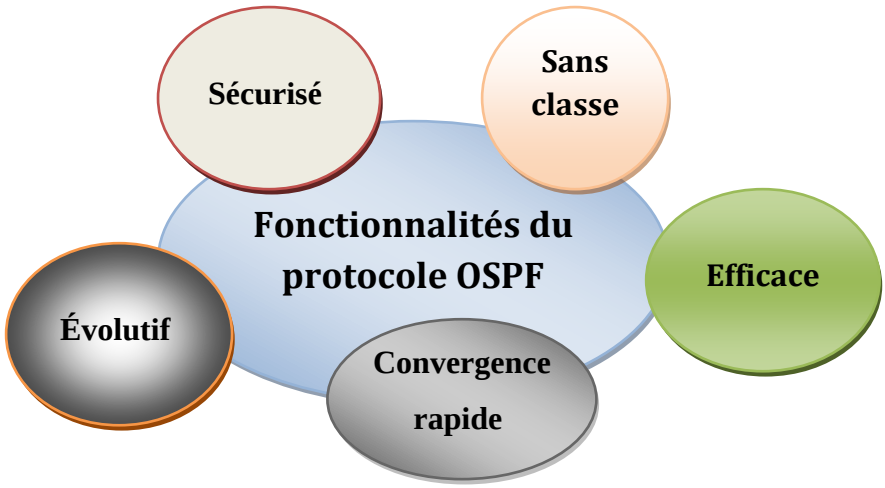


Figure 1.6 : Fonctionnalités du protocole OSPF

1.6.2 Fonctionnement général d'OSPF

Pour mettre à jour les informations de routage, les routeurs OSPF effectuent le processus de routage à état de liens générique qui suit afin d'atteindre un état de convergence :

- Établir les contiguïtés de voisinage (Figure 1.7) : Les routeurs compatibles OSPF doivent se reconnaître mutuellement sur le réseau avant de pouvoir partager des informations. Un routeur compatible OSPF envoie des paquets Hello à partir des interfaces compatibles OSPF pour déterminer si des voisins se trouvent sur ces liens. Si un voisin est présent, le routeur compatible OSPF tente d'établir une contiguïté de voisinage avec celui-ci.

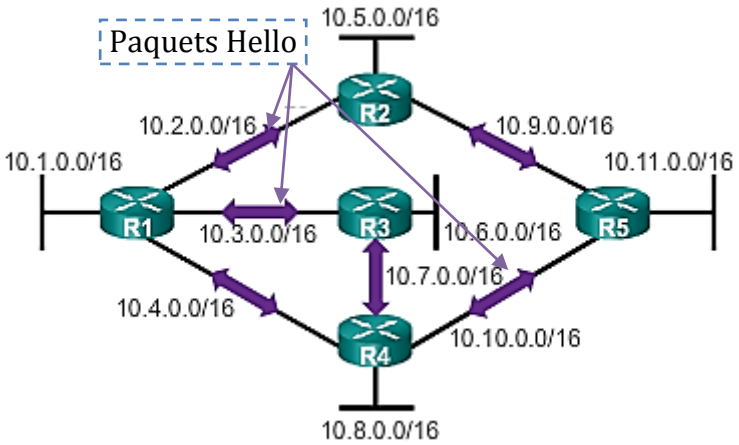


Figure 1.7 : les routeurs échangent des paquets Hello

- Échanger des paquets LSA (Link-State Advertisement) (Figure 1.8) : Une fois que les contiguïtés ont été établies, les routeurs échangent ensuite des paquets LSA. Les LSA contiennent l'état et le coût de chaque lien connecté directement. Les routeurs transmettent leurs LSA aux voisins contigus. Les voisins contigus recevant des LSA les diffusent

immédiatement aux autres voisins connectés directement, jusqu'à ce que tous les routeurs de la zone aient tous les LSA [25].

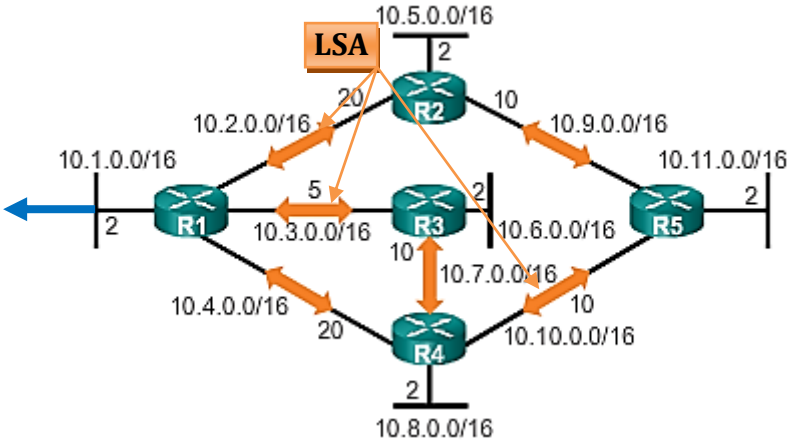


Figure 1.8 : les routeurs échangent des LSA

- Établir la table topologique (Figure 1.9) : Une fois les paquets LSA reçus, les routeurs compatibles OSPF créent la table topologique (LSDB) sur base des paquets LSA reçus. Cette base de données se retrouve alors à stocker toutes les informations relatives à la topologie du réseau [26].

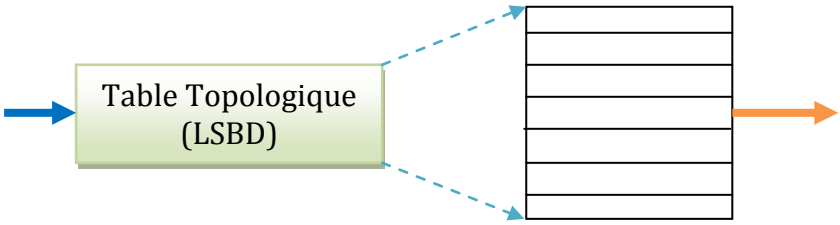


Figure 1.9 : Créer la base de données topologique

- Exécuter l'algorithme SPF (Figures1.10) : Les routeurs exécutent ensuite l'algorithme SPF. Les engrenages dans la figure sont utilisés pour indiquer le fonctionnement de l'algorithme SPF. L'algorithme SPF crée l'arborescence SPF.

  Le contenu de l'arborescence SPF de R1 est illustré dans le Tableau 1.3.

Figure 1.10 : exécuter de l'algorithme SPF et créer l'arborescence SPF

Destination	Chemin	Coût
10.5.0.0/16	R1→R2	22
10.6.0.0/16	R1→R3	7
10.7.0.0/16	R1→R3	15
10.8.0.0/16	R1→R3→R4	17
10.9.0.0/16	R1→R2	30
10.10.0.0/16	R1→R3→R4	25
10.11.0.0/16	R1→R3→R4→R5	27

Tableau 1.3 : Le contenu de l'arborescence

Les meilleurs chemins sont insérés dans la table de routage à partir de l'arborescence SPF. Les décisions de routage sont prises en fonction des entrées de la table de routage.

1.6.3. Encapsulation des messages OSPF

Les messages OSPF transmis sur un lien Ethernet contiennent les informations suivantes (Figure 1.11) :

- ✓ **En-tête de trame Ethernet de liaison de données** : Identifie les adresses MAC de multidiffusion de destination 01-00-5E-00-00-05 ou 01-00-5E-00-00-06.
- ✓ **En-tête de paquet IP** : Identifie le champ 89 du protocole IPv4 qui indique qu'il s'agit d'un paquet OSPF. Il identifie également l'une des deux adresses de multidiffusion OSPF : 224.0.0.5 ou 224.0.0.6 OSPF.
- ✓ **En-tête de paquet OSPF** - Identifie le type de paquet OSPF, l'ID du routeur et l'ID de zone.
- ✓ **Données spécifiques au type de paquet OSPF** : Contient des informations sur le type de paquet OSPF. Le contenu varie en fonction du type de paquet.

En-tête de trame Liaison de données	En-tête de paquet IP	En-tête de paquet OSPF	Base de données spécifique au type de paquet OSPF

Trame liaison de données
Adresse MAC de destination=Multidiffusion : 01-00-5E-00-00-05 ou 01-00-5E-00-00-06
Adresse MAC source =Adresse de l'interface d'envoi

Paquet IP
Adresse IP source =Adresse de l'interface d'envoi
Adresse IP de destination = Multidiffusion : 224.0.0.5 ou 224.0.06
Champ de protocole = 89 pour OSPF

En-tête OSPF
Code de type de paquet OSPF
ID de routeur et ID de zone

Type de paquets OSPF
0x01 Hello
0x02 description de base de données (DD)
0x03 LSR (Link-state-request)
0x04 LSU (Link-state-update)
0x05 LSAcK (Link-State Acknowledgement)

Figure 1.11 : Encapsulation des messages OSPF

1.6.4. Types de paquets OSPF

Le protocole OSPF utilise des paquets LSP (Link-State Packet) pour établir et maintenir des contiguïtés de voisinage, ainsi que pour échanger des mises à jour de routage [27].

Le Tableau 1.4 illustre les cinq types de paquets LSP utilisés par le protocole OSPF. Chacun d'eux a un objectif spécifique dans le processus de routage OSPF :

- **Type 1 : paquet Hello** : Etablit et maintient les informations de contiguïté (adjacency information) avec les voisins.
- **Type 2 : paquet DBD (Database Description)** : Décrit le contenu des bases de données d'état de liens (link-state database) des routeurs OSPF. La LSDB doit être identique sur tous les routeurs à état de liens au sein d'un secteur pour créer une arborescence SPF précise.
- **Type 3 : paquet LSR (Link-State Request)** : Demande des éléments spécifiques des bases de données d'état de liens (link-state database) des routeurs OSPF.

- **Type 4 : paquet LSU (Link-State Update)** : Transporte les link-sate advertisements, les LSA, aux routeurs voisins. Utilisé pour répondre aux paquets LSRs et pour annoncer de nouvelles informations.
- **Type 5 : paquet LSAck (Link-State Acknowledgment)** : Accusés de réception des LSA des voisins.

Type	Nom du paquet	Description
1	Hello	Découvre les voisins et crée des contiguïtés entre eux
2	Description de base de données	Vérifie la synchronisation de la base de données entre les routeurs
3	LSR (Link-State Request)	Demande des enregistrements d'état de liens spécifiques d'un routeur à un autre
4	LSU (Link-State Update)	Envoie les enregistrements d'état de liens spécifiquement demandés
5	LSAcK (Link-State Acknowledgment)	Reconnait les autres types de paquet

Tableau 1.4 : Description de paquet OSPF

1.6.5. Paquet Hello

Le paquet de type 1 du protocole OSPF correspond au paquet Hello. Les paquets Hello sont utilisés pour :

- Découvrir les routeurs voisins OSPF et établir des contiguïtés entre eux.
- Annoncer les paramètres sur lesquels les deux routeurs doivent s'accorder pour devenir voisins
- Définir le routeur désigné (DR) et le routeur désigné de secours (BDR) sur les réseaux à accès multiple, de type Ethernet et Frame Relay.

La figure 1.12 affiche les champs contenus dans les paquets Hello :

- **Type** : Identifie le type de paquet. Un (1) indique un paquet Hello. La valeur 2 identifie un paquet DBD, 3 un paquet LSR, 4 un paquet LSU et 5 un paquet LSAck.
- **ID du routeur** : Valeur 32 bits exprimée en notation décimale à point utilisée pour identifier le routeur d'origine de façon unique.
- **ID de zone** : Zone d'où provient le paquet.
- **Masque réseau** : Masque de sous-réseau associé à l'interface émettrice.
- **Intervalle Hello** : L'intervalle Hello spécifie le délai, en secondes, entre les paquets Hello qu'un routeur envoie sur une interface OSPF. L'intervalle Hello par défaut sur des réseaux à accès multiple est de 10 secondes.
- **Priorité du routeur :** Utilisé dans une sélection DR/BDR. La priorité par défaut pour tous les routeurs OSPF correspond à 1, mais elle peut être changée manuellement en une valeur

comprise entre 0 et 255. Plus la valeur est élevée, plus le routeur devient le routeur désigné sur le lien.

- **Intervalle Dead** : (Intervalle d'inactivité) est le nombre de secondes pendant lesquelles les paquets Hello d'un routeur n'ont pas été vus avant que ses voisins ne déclarent le routeur OSPF inactif.
- **Routeur désigné (DR)** : ID du routeur désigné.
- **Routeur désigné de secours (BDR)** : ID du routeur désigné de secours.
- **Liste des voisins** : Liste qui identifie les ID de routeur de tous les routeurs adjacents.

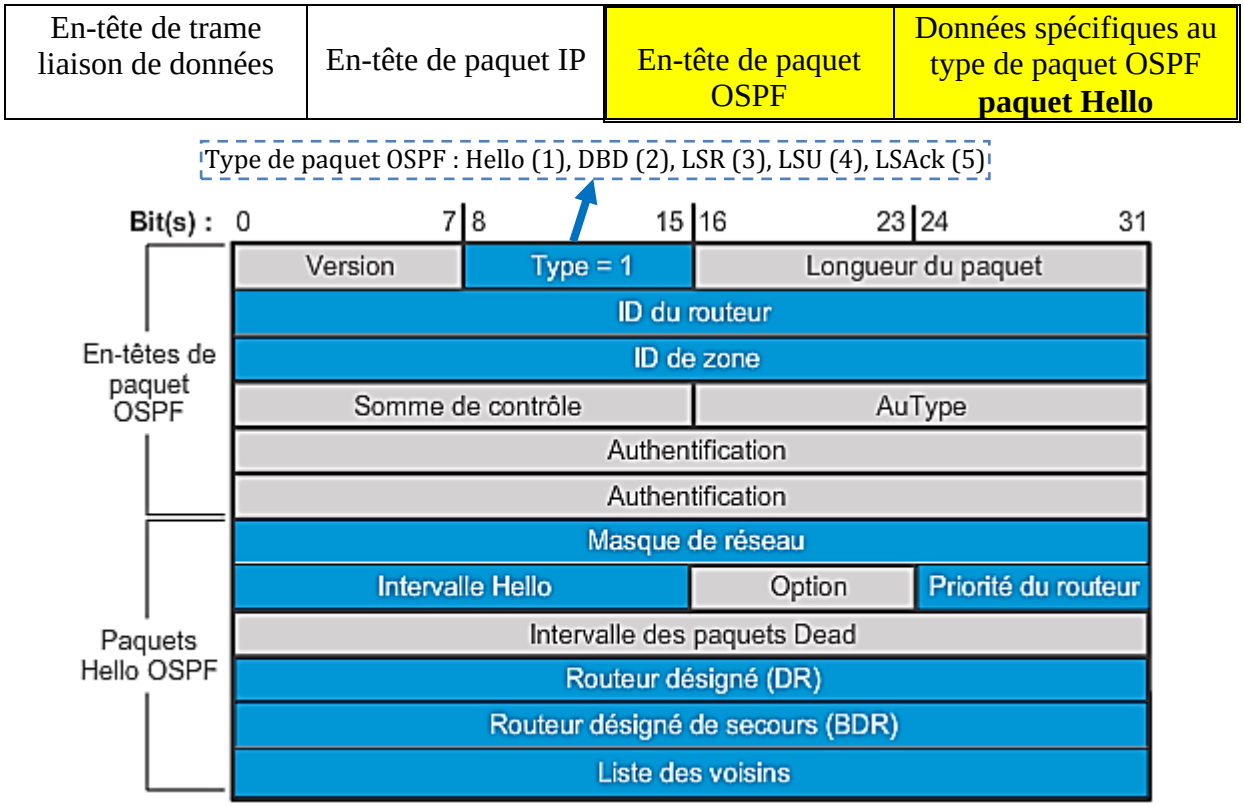


Figure 1.12: Contenu du paquet Hello OSPF

A. Intervalle du paquet Hello

Les paquets Hello du protocole OSPF sont transmis à l'adresse multicast 224.0.0.5 toutes les :

- 10 secondes (valeur par défaut dans les réseaux à accès multiple et point à point)
- 30 secondes (valeur par défaut sur les réseaux à accès multiple sans diffusion NBMA ; par exemple, à relais de trames)

L'intervalle Dead correspond à l'écart de temps pendant lequel le routeur attend de recevoir un paquet Hello avant de déclarer le voisin hors service. Si l'intervalle Dead arrive à échéance avant que les routeurs ne reçoivent un paquet Hello, le protocole OSPF supprime le voisin de sa

LSDB. Le routeur diffuse vers la LSDB les informations concernant le voisin hors service vers toutes les interfaces compatibles OSPF [28].

B. DR et BDR OSPF

Pourquoi une sélection du routeur DR et du routeur BDR est-elle nécessaire ?

Les LSA sur les réseaux à accès multiple peuvent présenter deux difficultés pour OSPF :

- **Création de contiguïtés multiples** : Les réseaux Ethernet pourraient éventuellement interconnecter de nombreux routeurs OSPF via un lien commun. La création de contiguïtés avec chaque routeur est inutile et non souhaitée. Elle se traduirait par un nombre excessif de paquets LSA circulant entre les routeurs du même réseau.
- **Diffusion massive de paquets LSA** : Les routeurs à état de liens diffusent leurs paquets LSA chaque fois que le protocole OSPF est initialisé, ou lorsqu'une modification topologique se produit. Cette diffusion peut devenir excessive.

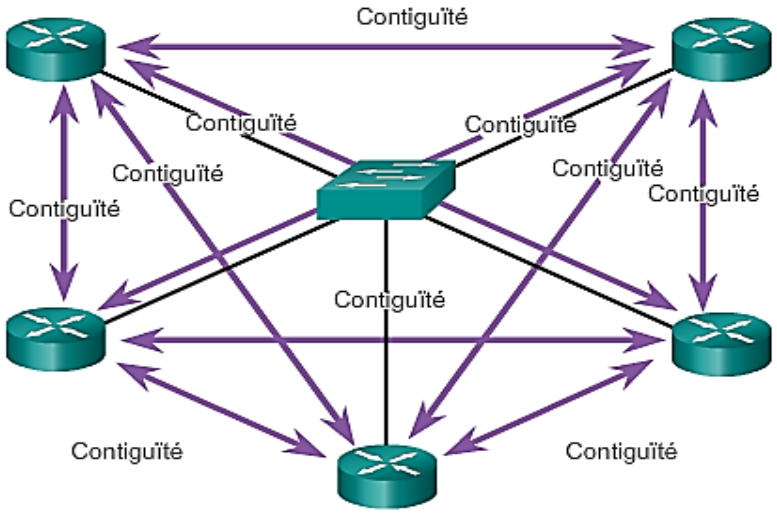


Figure 1.13 création de contiguïtés avec chaque voisin

Pour mieux appréhender le problème des contiguïtés multiples, nous devons étudier une formule :

Pour un nombre quelconque de routeurs (indiqué par n) sur un réseau à accès multiples, il y a $n\,(n-1)/2$ contiguïtés.

La Figure 1.13 illustre une topologie à cinq routeurs, tous rattachés au même réseau à accès multiple Ethernet. S'il n'existe aucun mécanisme permettant de réduire le nombre de contiguïtés, ces routeurs formeront, ensemble, 10 contiguïtés : $5\,(5-1)/2 = 10$

Cela peut sembler peu, mais à mesure que des routeurs sont ajoutés au réseau, le nombre de contiguïtés augmente de façon considérable, comme illustré dans le tableau 1.5.

Routeurs	Contiguïtés
n/5	[n (n-1) / 2] /10
10	45
20	190
100	4 950

Tableau 1.5 : plus de routeurs = plus de contiguïtés

1.6.6. Les zones OSPF (unique et zone multiple)

Pour une efficacité et une évolutivité supérieures, le protocole OSPF prend en charge le routage hiérarchique à l'aide de zones. Les zones OSPF permettent d'isoler des parties du réseau afin de diminuer la taille de la topologie réseau à mémoriser sur chaque routeur.

Le protocole OSPF peut être mise en œuvre de deux manières différentes :

- **OSPF à zone unique** : Dans la figure 1.14, tous les routeurs sont situés dans une zone appelée zone fédératrice (zone 0).

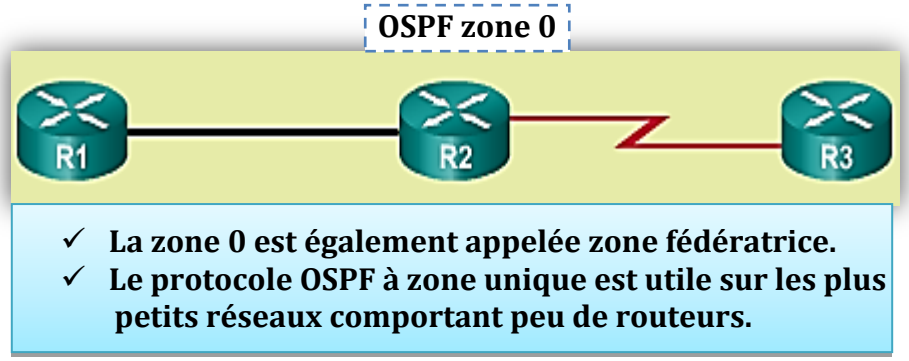


Figure 1.14 : protocole OSPF à zone unique

- **OSPF multizone** - Dans la figure 1.15, le protocole OSPF est mis en œuvre à l'aide de plusieurs zones, de façon hiérarchique. Toutes les zones doivent se connecter à la zone de réseau fédérateur (zone 0). Les routeurs reliant les zones sont appelés routeurs ABR (Area Border Router).

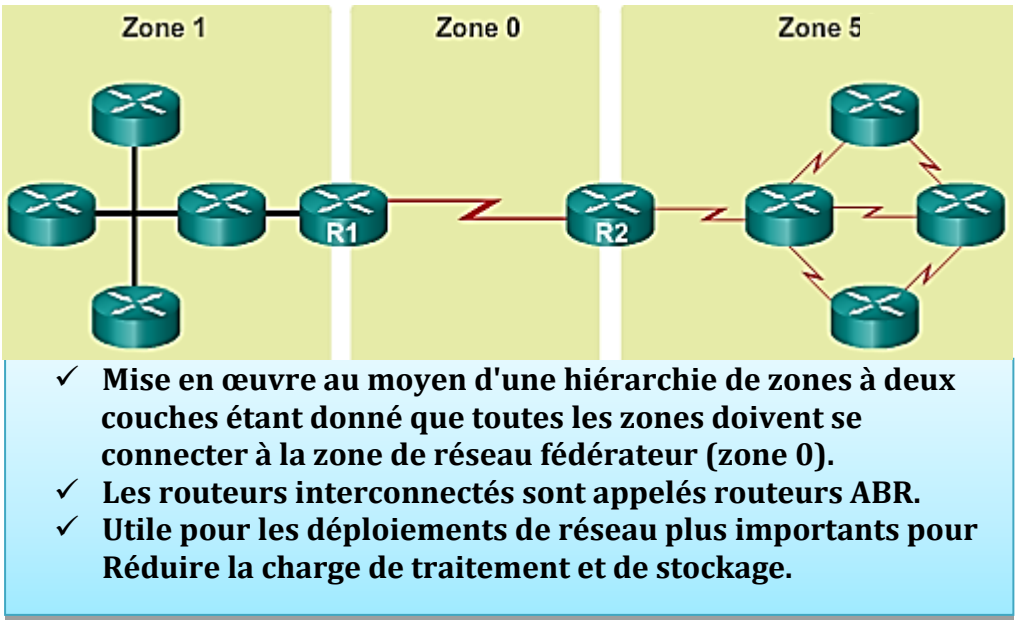


Figure 1.15 : protocole OSPF à zone multiple

Avec le protocole OSPF multizone, OSPF peut diviser un grand système autonome (AS) en zones plus petites, pour prendre en charge le routage hiérarchique. Avec le routage hiérarchique, le routage s'effectue toujours entre les zones (routage interzone), alors que la plupart des opérations de routage exigeant beaucoup de temps du processeur, comme le recalcule de la base de données, sont conservées dans une zone.

Par exemple, chaque fois qu'un routeur reçoit de nouvelles informations relatives à une modification topologique dans la zone (notamment l'ajout, la suppression ou la modification d'un lien), le routeur doit relancer l'algorithme SPF, créer une nouvelle arborescence SPF et mettre à jour la table de routage. L'algorithme SPF est exigeant en temps processeur et le temps qu'il prend pour le calcul dépend de la taille de la zone.

Remarque : les modifications de topologie sont distribuées aux routeurs des autres zones dans un format de vecteur de distance. En d'autres termes, ces routeurs mettent uniquement à jour leurs tables de routage et ne doivent pas réexécuter l'algorithme SPF.

Un nombre excessif de routeurs dans une zone rendrait les LSDB très volumineuses et augmenterait la charge sur le processeur. Par conséquent, l'organisation des routeurs en zones partitionne efficacement une base de données potentiellement volumineuse en bases de données plus petites et plus faciles à gérer.

Le protocole OSPF multizone présente les avantages suivants [29] : (La figure 1.16 illustre ces avantages)

- **Réduction de la taille des tables de routage** : Moins d'entrées dans la table de routage parce que les adresses réseau peuvent être récapitulées entre les zones. La récapitulation de route n'est pas activée par défaut.
- **Réduction de la surcharge liée aux mises à jour d'état de liens** : Réduit les exigences de traitement et de mémoire.
- **Réduction de la fréquence des calculs SPF** : Recherche l'impact d'une modification topologique au sein d'une zone. Par exemple, l'impact des mises à jour de routage est limité parce que l'inondation des paquets LSA s'arrête à la limite de zone.

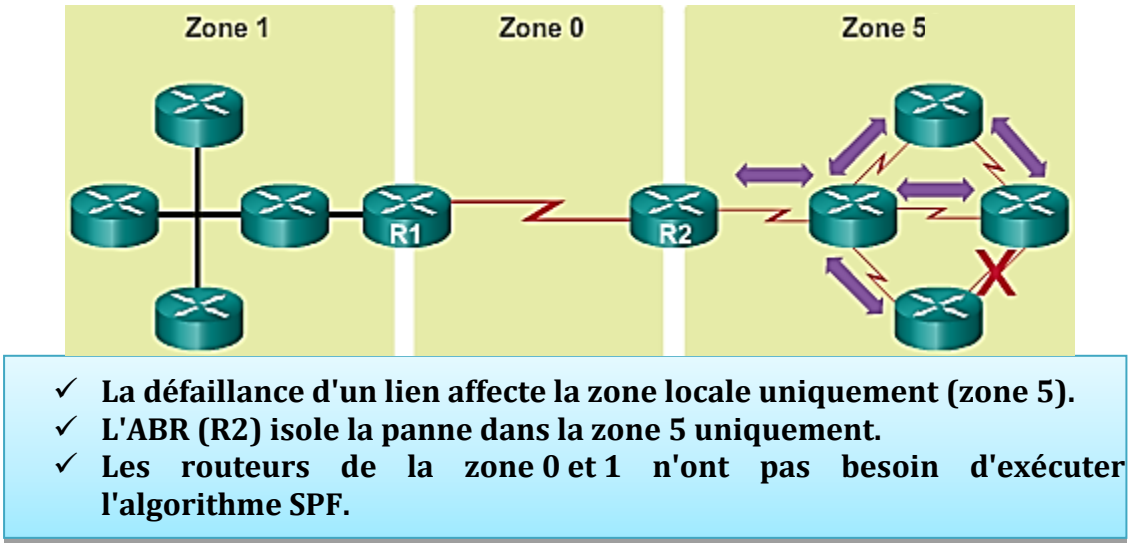


Figure 1.16 : la modification du lien affecte uniquement la zone locale

Par exemple, R2 est un routeur ABR pour la zone 5. En tant que routeur ABR, il récapitulerait les routes de la zone 5 dans la zone 0. Lorsque l'un des liens récapitulés échoue, les paquets LSA sont échangés au sein de la zone 5 uniquement. Les routeurs de la zone 5 doivent exécuter de nouveau l'algorithme SPF pour identifier les meilleures routes. Cependant, les routeurs de la zone 0 et de la zone 1 ne reçoivent aucune mise à jour ; par conséquent, ils n'exécutent pas l'algorithme SPF.

1.7 Protocoles dérivés de protocole OSPF (QOSPF)

QOSPF (Quality Of Service Path First) [30], [31] est une extension d'OSPF. Combiné à un protocole de réservation, ce protocole de routage avec qualité de service permet d'annoncer à tous les routeurs, la capacité des liens à supporter des contraintes de qualité de service. QOSPF

se base sur l'utilisation de deux messages, le RES-LSA (Link Resource Advertisement message) et le RRA (Resource Reservation Advertisement message).

Le but des messages RES-LSA est d'informer l'ensemble des routeurs de l'état des liens. Ces informations sont utilisées pour calculer les chemins. Ce type de message est ainsi utilisé lorsqu'un nouveau routeur est ajouté au réseau ou lorsque la charge d'un lien varie. Le rôle d'un message RRA est d'indiquer les ressources utilisées par un flux (ou plutôt réservées pour ce flux). Chaque routeur est averti des ressources utilisées et du chemin emprunté par chaque flux. Le nombre de messages RRA peut donc devenir très élevé si le nombre de flux augmente, ce qui pose un problème d'échelle. Cependant, on peut y remédier en effectuant un routage explicite : il s'agit de pré-calcul de manière centralisée de la route permettant de satisfaire les contraintes de qualité de service, et d'imposer ce chemin à chaque routeur. [32]

Typiquement, ce calcul est effectué par la source (routage par la source), QOSPF n'est pas autorisé à remplacer un itinéraire existant sous prétexte qu'un chemin s'avère meilleur. C'est donc la stabilité qui est privilégiée par rapport à la performance immédiate.

1.8 Conclusion

Dans ce chapitre, nous avons passé en revue les différentes techniques permettant de garantir le processus de routage dans les réseaux de télécommunication. Ainsi, nous avons expliqué qu'OSPF est un protocole relativement complexe, nécessitant non seulement des ressources matérielles importantes, mais également de bonnes connaissances et une bonne maîtrise de son fonctionnement avant de pouvoir être implémenté. En dépit de cette complexité de mise en œuvre, il augmentera sensiblement les performances, la stabilité et la fiabilité du réseau. Le tableau suivant nous donne une conclusion sur le protocole OSPF.

Fonctionnalité	Link State	Vecteur de distance
Temps de convergence	Rapide	Lent à cause de la détection des boucles
Suppression des boucles	Inhérent au protocole	Nécessite des mécanismes spécifiques
Besoin en Mémoire et CPU	Peut-être important	Faible
Nécessite des efforts de Conception pour les grands réseaux	Oui	Non

Chapitre 2

L'ALGORITHME DE DIJKSTRA

2.1. Introduction

La gestion de l'acheminement de données ou le routage consiste à assurer une stratégie qui doit permettre à n'importe quel moment, la connexion entre n'importe quelle paire de nœuds appartenant à un réseau. La diffusion est utilisée afin d'inonder le réseau et les informations sont alors commutées de proche en proche par les mobiles intermédiaires.

À l'origine, les réseaux étaient tous de taille très limitée et il était donc possible de configurer manuellement la route pour chaque destination. Mais très vite, la taille des réseaux va qu'en s'accroissant, il a fallu automatiser cette tâche à l'aide d'algorithmes de routage.

Certains algorithmes de routage ont été définis très tôt, mais restent des méthodes solides qui ont fait leurs preuves qui sont encore utilisées aujourd'hui, et qui gardent une influence très importante sur les travaux réalisés dans ce domaine [33] [34] [35].

Les notions de routage et d'acheminement sont différentes, bien que fortement liées : le routage consiste à trouver une route pour une destination tandis que l'acheminement consiste à transmettre le paquet d'un routeur à l'autre sur la route trouvée. L'acheminement est donc une phase postérieure au routage, mais certains algorithmes l'utilisent néanmoins de manière rétroactive sur le routage. Par exemple, l'acheminement peut permettre de valider une route, et en cas de problème, de réenclencher l'algorithme de routage [6].

2.2 Théorie des graphes

Les équipements physiques tels que les terminaux, multiplexeurs, concentrateurs, liaisons de communication, ordinateurs et autres qui composent un réseau de troisième génération peuvent être représentés par des nœuds et des arcs, concepts fondamentaux sur lesquels repose la théorie des graphes [7], nous allons définir certains concepts de cette théorie qui sert de base à l'analyse et à la création topologique des réseaux.

2.2.1 Graphe

On appelle graphe un ensemble de nœuds (ou sommets) reliés entre eux par des arcs (ou arêtes). Si l'on désigne par N l'ensemble des nœuds et par A l'ensemble des arcs, on peut noter que G = (N, A) est le graphe qui en résulte. Pour un ensemble N de n nœuds, le nombre maximum d'arcs est m tel que le m_{max}= n (n — 1) /2.

2.2.2 Arc orienté

Tout arc qui comporte une origine et une destination non interchangeable est dit orienté, la destination étant généralement indiquée par une flèche à l'extrémité correspondante de l'arc.

Dans le cas contraire, l'arc est dit non orienté.

Un arc orienté sert à modéliser une liaison de communication unidirectionnelle (simplex), alors qu'un arc non orienté représente une liaison bidirectionnelle (duplex) [38].

2.2.3 Graphe orienté et graphe non orienté

On dit qu'un graphe est orienté lorsque tous ses arcs le sont, lorsque les arcs d'un graphe sont non orientés, le graphe est dit non orienté.

2.2.4 Architecture et degré d'un graphe

On dit d'un arc est incident à un nœud si cet arc admet ce nœud comme origine ou destination. On appelle degré d'incidence d'un nœud le nombre d'arcs incidents à ce nœud. On désigne par degré d'un graphe le degré d'incidence du nœud de plus faible degré [39].

2.2.5 Notion de chemin

On appelle chemin de i→j entre deux nœuds i et j d'un graphe la suite d'arcs permettant, à partir du nœud i, d'atteindre le nœud j.

Chaque arc est muni d'une longueur qui n'est autre que la distance euclidienne entre ses nœuds origine et destination. En conséquence, on désigne par longueur d'un chemin la somme des longueurs d'arcs qui composent celui-ci.

Un chemin dont l'origine se confond à la destination constitue un cycle. On l'appelle également chemin fermé ou circuit. [40].

2.2.6 Connectivité

La connectivité d'un nœud est le nombre d'arc connectant ce nœud à des nœuds adjacents.

Dans la figure ci-dessous le nœud X a une connectivité de 3 :

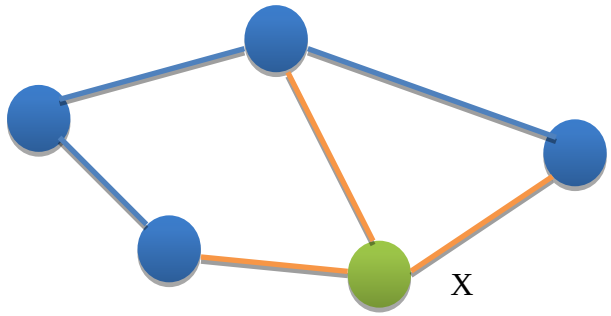


Figure 2.1 Connectivité d'un nœud

La connectivité traduit le degré de maillage de réseau et, de ce fait, est un moyen intermédiaire d'apprécier sa fiabilité potentielle.

Pour vérifier qu'un graphe est connecté, c'est-à-dire que chaque nœud peut atteindre un autre par un chemin direct ou via un autre nœud, on peut appliquer deux méthodes :

✓ La première consiste à vérifier que, pour chaque couple de nœuds (X, Y) il existe un chemin permettant d'atteindre Y depuis X. Cette méthode a l'avantage de la simplicité mais présente l'inconvénient d'avoir un temps d'exécution important.

En effet, l'algorithme passe un temps important à redécouvrir des chemins qu'il a déjà obtenus pour les couples (X,Y) précédents.

✓ La seconde est fondée sur l'assertion, selon laquelle un nœud connecté à un autre nœud déjà connecté. Cet algorithme a l'avantage d'une plus grande vitesse d'exécution lorsque la taille de graphe commence à devenir un peu importante.

2.2.7 Arbre sous-tendant minimum

On désigne par arbre ou arborescence un graphe connexe sans cycle. Le degré de connexité d'un tel graphe est donc égal à 1. Un arbre est dit sous-tendant minimum lorsqu'il relie tous les nœuds considérés au moyen d'arcs dont la somme totale des longueurs est le plus petit possible. En considérant celle-ci comme un poids, on peut également utiliser la notion plus générale d'arbre de poids minimum. La conception d'un réseau nécessite, pour être menée à bien, de disposer d'un graphe pour la réalisation du plan de routage et d'une matrice de bout en bout réduite déterminant les flux de données [41].

2.2.8 Choix d'un graphe de réseau

Concevoir un graphe, c'est relier les nœuds d'un réseau par des arcs pour pouvoir ensuite y projeter les flux et effectuer le dimensionnement des nœuds et des liaisons.

Dans le cas où on trouverait un flux de bout en bout important entre deux nœuds, il est nécessaire de placer un arc de façon à permettre au trafic d'y aller directement plutôt que de passer par des arcs et des nœuds intermédiaires ou bien utiliser une fonction coût qui prend en charge la recherche du meilleur chemin en terme de bande passante, délai, débit, taux de perte...

En fait, la connaissance d'un graphe de réseau permet de caractériser certaines propriétés liées à la fiabilité, telle que la connectivité minimale, la probabilité de déconnexion et le dénombrement des chemins disjoints. Les autres propriétés telles que les performances, l'efficacité, le facteur de blocage et le flux maximal impliquant avoir défini totalement le réseau (graphe, capacité, routage, etc.) [42].

2.2.9 Projection des flux et routage

La détermination des capacités et des flux qui parcourent un réseau nécessite un dimensionnement de ses composants. Ce dimensionnement est réalisé par une projection et un routage des flux sur le graphe.

La projection des flux sur le graphe consiste à produire la matrice de flux physique χ, c'est-à-dire à établir quelle valeur du flux est assignée à chaque arc et à chaque nœud, en partant de la matrice de bout en bout réduite φ (qui définit le trafic à écouler entre chaque nœud du réseau cœur), du graphe et des règles (ou plan) de routage.

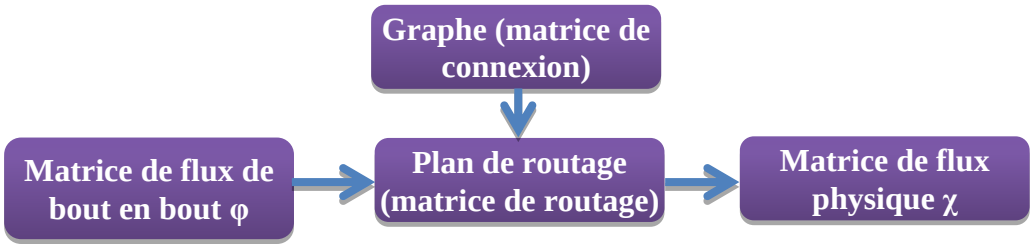


Figure 2.2 Projection du flux sur le graphe

2.1.10 Routage

Le routage consiste à déterminer, pour chacun des flux utiles de bout en bout, quelle sera la séquence d'arcs empruntés dans le graphe considéré. Il existe deux types de routages : Routage au plus court chemin : Le routage est de type simple (mono chemin), c'est-à-dire qu'un flux (i,j) ne suit qu'une seule route. La construction de sa matrice nécessite une méthode systématique présentée dans l'algorithme du plus court chemin (Shortest path de Dijkstra). Cet algorithme est utilisé pour définir le plan de routage statique.

Routage multi chemin : Le routage sur la base du chemin le plus court a l'intérêt de l'efficacité mais présente aussi l'inconvénient de ne pas tirer pleinement parti de la bande passante disponible. En assurant le trafic entre les différents chemins offerts, le routage multi chemin permet d'améliorer la fiabilité du réseau, de réduire la charge des liaisons, d'améliorer les performances, d'utiliser pleinement la bande passante disponible et par conséquent de réduire le coût (voir figure 2.3)

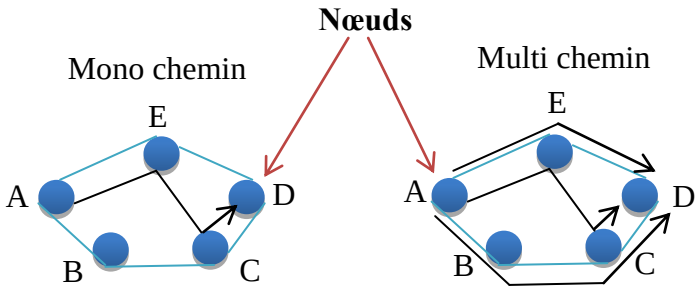


Figure 2.3 Routage mono chemin et multi chemin

2.3. L'algorithme de Dijkstra

Créé par l'informaticien Néerlandais Edsger Dijkstra en 1956 et publié en 1959, il a proposé un algorithme cité par tous. Il y propose une manière de trouver le chemin de poids le plus faible entre deux points d'un graphe non-orienté pondéré dont les arêtes ont un poids positif ou nul. Peu de temps après, en 1960, Whiting et Hillier ont présenté le même algorithme, en suggérant que la méthode pouvait aisément être appliquée aux graphes orientés, produisant un

arbre de plus court chemin. Cet algorithme est souvent utilisé dans le routage et un sous-programme dans d'autres algorithmes de graphe [43].

Pour un sommet source donné (nœud) dans le graphe, l'algorithme trouve le chemin à coût le plus faible (i.e. le plus court chemin) entre ce sommet et tous les autres sommets. Il peut également être utilisé pour trouver des coûts de chemins les plus courts à partir d'un seul sommet à un sommet unique de destination en arrêtant l'algorithme une fois que le plus court chemin vers le sommet de la destination a été déterminé. Par conséquent, l'algorithme de Dijkstra (Shortest Path First) est largement utilisé dans les protocoles de routage du réseau, notamment IS-IS et OSPF (Open Shortest Path First) [44]. Soient n le nombre de nœuds et a le nombre d'arcs. Le coût total pour l'algorithme de Dijkstra est : O (|a| + |n| ln |n|) [45].

2.3.1 Le principe de l'algorithme

L'algorithme de Dijkstra est un algorithme de type glouton : à chaque nouvelle étape, on traite un nouveau sommet. Reste à définir le choix du sommet à traiter, et le traitement à lui infliger [46].

Tout au long du calcul, on met à jour deux ensembles :

C Ensemble des sommets restants à visiter (au départ C=S\ {source}).

D Ensemble des sommets pour lesquels on connaît déjà leur plus petite distance à la source (au départ, D= {source}).

L'algorithme se termine quand C est vide.

Pour chaque sommet s dans D, on conservera :

- Dans un tableau *distances*, le poids du plus court chemin jusqu'à la source,
- Dans un tableau parcours, le sommet p qui le précède dans un plus court chemin de la source à s.

Ainsi, pour retrouver un chemin le plus court, il suffira de remonter de prédécesseur en prédécesseur jusqu'à la source, ce qui pourra se faire grâce à un unique appel récursif (beaucoup moins coûteux que dans le cas de Floyd).

2.3.2 INITIALISATION

Au début de l'algorithme, le chemin le plus court connu entre la source et chacun des sommets est le chemin direct, avec une arête de poids infini s'il n'y a pas de liaison entre les deux sommets.

On initialise donc le tableau distances par les poids des arêtes reliant la source à chacun des sommets, et le tableau parcours par *source* pour tous les sommets.

2.3.3 I^{ère} ÉTAPE

On suppose avoir déjà traité *i* sommets, *parcours* et *distances* contiennent respectivement les poids et le prédécesseur des plus courts chemins pour chacun des sommets déjà traités.
Soit *s* le sommet de *C* réalisant le minimum de *distances[s]*.

- On supprime *s* de *C* et on l'ajoute à *D*.
- Reste à mettre à jour les tableaux *distances* et *parcours* pour les sommets *t* reliés directement à *s* par une arête, comme suit.

Si *distances[s] + F(s,t) < distances[t]*,
Alors on remplace :

- *Distances [t]* par *distances[s] + F(s,t)*,
- *parcours[t]* par *s*.

2.3.4 (N-2)^{ème} ÉTAPE

Au départ, il y a *(n-1)* sommets à visiter, mais la dernière étape est inutile (elle n'apporte rien). Dès la *(n-2)*^{ème} étape, *distances* et *parcours* contiennent toute l'information nécessaire pour trouver les plus courts chemins.

2.3.5 EVALUATION

Si n est le nombre de sommets du graphe, et a son nombre d'arêtes, alors l'algorithme de Dijkstra renvoie le résultat au bout de O ($a^2 \ln|n|$) opérations.
Pour un même résultat, Floyd nécessite O (n3) opérations.

2.4. Description de l'algorithme de Dijkstra

L'algorithme de Dijkstra classique, reposant sur le principe de la création d'étiquettes [47]. Ce choix est justifié étant donné que la recherche de plus courts chemins est basée sur la minimisation du nombre de sauts, où le coût d'un arc est égal à un. De plus, le principe d'optimalité retenu ne stipule qu'un plus court chemin est constitué de plus courts chemins élémentaires. Par conséquent, la recherche du plus court chemin consiste à explorer à chaque itération le nœud ayant la plus petite distance.
L'algorithme de Dijkstra classique comprend trois étapes :

- **Etape d'initialisation :** A l'initialisation de l'algorithme, l'ensemble des étiquettes éligibles X ne contient que l'étiquette du nœud source. La distance cumulée de cette étiquette ainsi que le nombre de chemins P sont nuls.

- **Etape 1 :** Dans cette étape, on cherche dans l'ensemble X l'étiquette qui présente laplus courte distance cumulée. Celle-ci est alors retirée de X. Si le nœud associé à l'étiquette sélectionnée correspond au nœud destination alors ce nœud est ajouté au chemin sinon on passe à l'étape 2.
- **Etape 2 :** Dans cette étape, les arcs partant du nœud correspondant à l'étiquette sélectionnée dans l'étape 2 sont explorés. Les étiquettes des nœuds situés à l'autre extrémité sont ajoutées à l'ensemble des étiquettes éligibles X et la distance cumulée correspondant à chaque étiquette est alors calculée.

NOTA :

La recherche d'une étiquette dépend du critère utilisé. Par exemple, si ce dernier concerne la recherche de la meilleure bande passante sur un chemin, l'étiquette sélectionnée correspond alors à celle ayant la plus grande distance cumulée. Par contre si, le critère utilisé concerne la minimisation du nombre de sauts alors l'étiquette sélectionnée est celle dont la distance cumulée est la plus courte. Dans l'étape 2, pour pouvoir remonter tout le chemin jusqu'à la source, il est nécessaire de sauvegarder le nœud précédant chaque nœud étiqueté.

** N – L'ensemble des nœuds du réseau*

** X – Liste des nœuds "éligibles"*

** $Count_i$ – Nombre de chemins déjà calculés entre s et i*

** elem – Nombre d'étiquettes assignées*

** P – Liste des chemins de s à t*

** K – Nombre de chemins à calculer*

/ Initialisation */*

$count_i$ = 0 pour tout i appartenant à N

elem = 1 (Il n'y a qu'une seule étiquette pour l'instant : celle associée au nœud source (s))

(Associe l'étiquette et le nœud)

h(elem) = s

$h^{-1}(s)$ = {elem}

$distance_{elem}$ = 0

X = {elem}

P^K = 0 / Aucun chemin n'a encore été calculé */*

While ($count_t$< K and X != 0) / Tant qu'on n'a pas calculé les K chemins et qu'il existe encore des candidats */*

```
begin
/* ETAPE 1 */
    (Trouver l'étiquette éligible présentant la plus courte distance)
    k = element de X tel que distance_k <= distance_x pour tout x de X
    X = X – {k}                    (Retirer l'étiquette de la liste des candidats)
    i = h(k)                       (Trouver le nœud associé à l'étiquette)
    count_i = count_i + 1          (Incrémenter le compteur de chemins)
    if (i == t)alors               (Si on est arrivé à destination)
        begin                      (Ajouter le nouveau chemin)
            p = chemin de 1 à k
            P^K = P^K U {h(p)}
        end
/* ETAPE 2*/
    If (count_i <= K) alors        (S'il n'y a pas trop de chemins passant par ce nœud)
        begin
            for eacharc(i,j) Є i   (Pour chaque lien appartenant à ce nœud)
            begin
                elem = elem + 1    (Ajouter le nœud situé à l'autre extrémité
                                    de l'arc, dans la liste des candidats)
                distance_elem = distance_k + c_ij   (Sauvegarde des informations de
                                                     la nouvelle étiquette)
                precedent_elem = k
                h(elem) = j        (Pour retrouver le nœud auquel
                                    appartient l'étiquette)
                h^-1(j) = h^-1(j) U {elem}
                X = X U {elem}     (Ajouter l'étiquette à la liste des
                                    candidats)
            end
        end
end
```

Figure 2.4 : Algorithme de Dijkstra généralisé [32]

Exemple :

Considérons un réseau informatique défini par le graphe valué Γ = (X, A) et dont la représentation est faite ci-dessous :

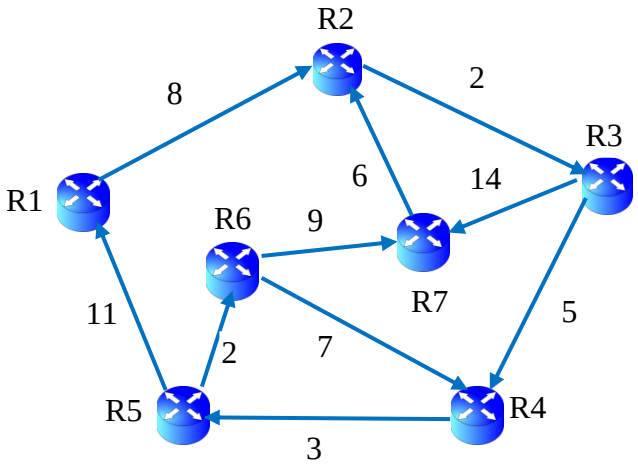


Figure 2.5 : graphe1

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**

Tableau 2.1: table 1

On attribue au sommet choisi (ici R3) la marque 0, et à tous les autres sommets une marque initiale infinie. En pratique, les ordinateurs ne connaissant pas les ensembles infinis, il suffit de donner à chaque sommet une marque initiale égale à la somme des évaluations de tous les arcs du graphe, (table ci-dessus).

R3 conserve sa marque 0. On remplace la marque des successeurs k de 1 par la valeur de l'arc (R3 ; k) :

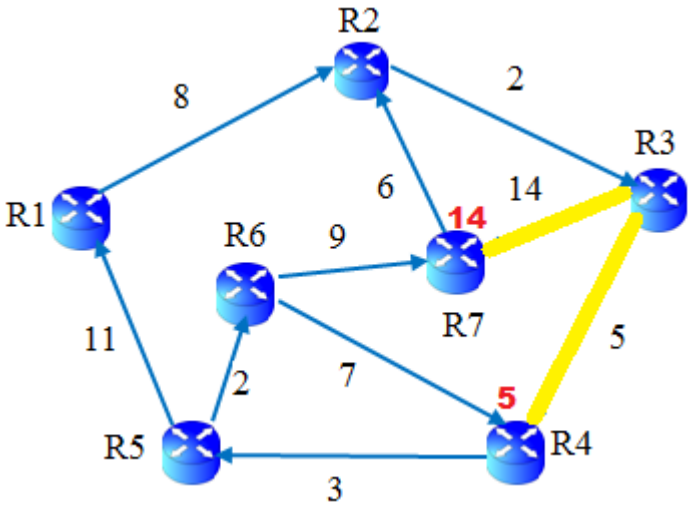


Figure2.6 : graphe 2

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	

Tableau 2.2: Table 2

3) Dans l'ensemble des sommets X, {R3} on choisit un (car il peut ne pas être unique) sommet de plus petite marque (ici R4, de marque $y_{R4} = 5$) et on attribue à tous ses successeurs k la marque Inf $\{y_k, y_{R4} + c_{R4}k\}$. On obtient donc :

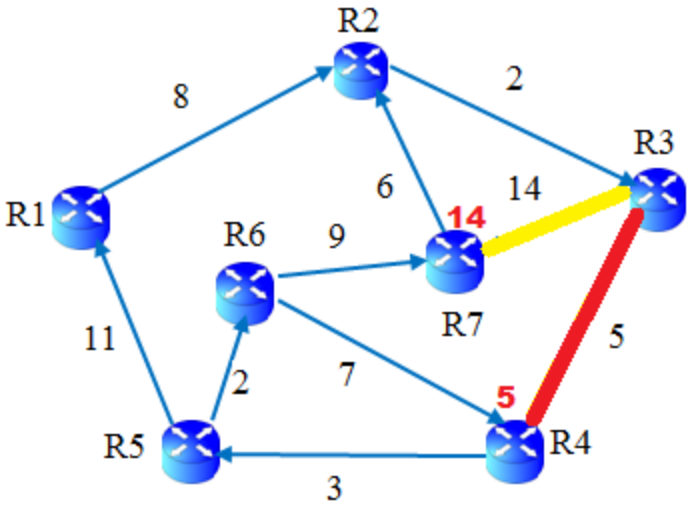


Figure2.7 : graphe3

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**

Tableau 2.3: Table 3

4) Dans l'ensemble des sommets X – {R3, R4} on choisit un sommet de plus petite marque (ici R5, de marque $y_{R5} = 8$) et on attribue à tous ses successeurs *k* la marque *Inf* $\{y_k, y_{R5}, c_{R5}K\}$. On obtient donc :

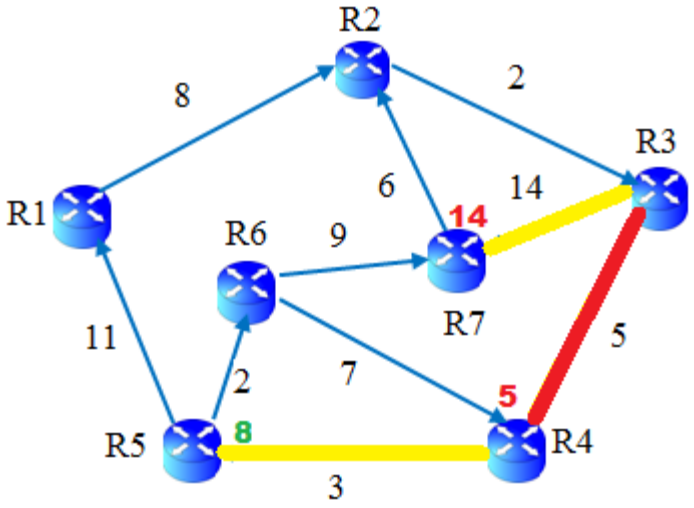


Figure 2.8 : graphe 4

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	

Tableau 2.4: Table 4

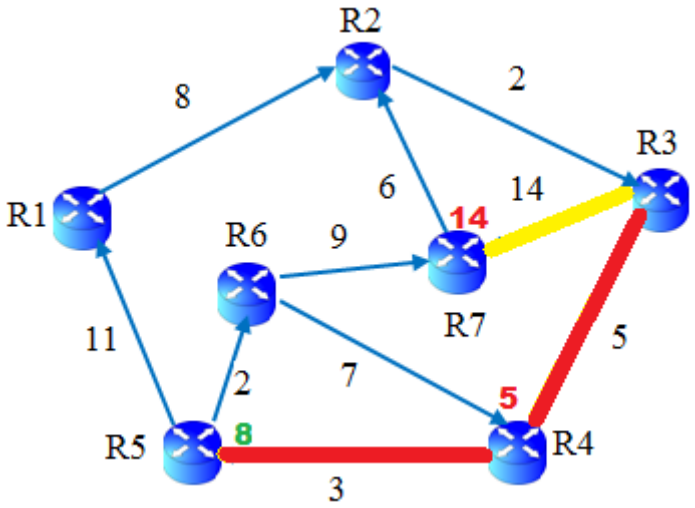


Figure 2.9 : graphe 5

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**

Tableau 2.5: Table 5

On réitère le procédé : dans l'ensemble des sommets X – {R3, R4, R5} on choisit un sommet de plus petite marque (ici R6, de marque y = 10) et on attribue à tous ses successeurs *k* la marque Inf {yB*k*, y_{R6} + cR6k}.

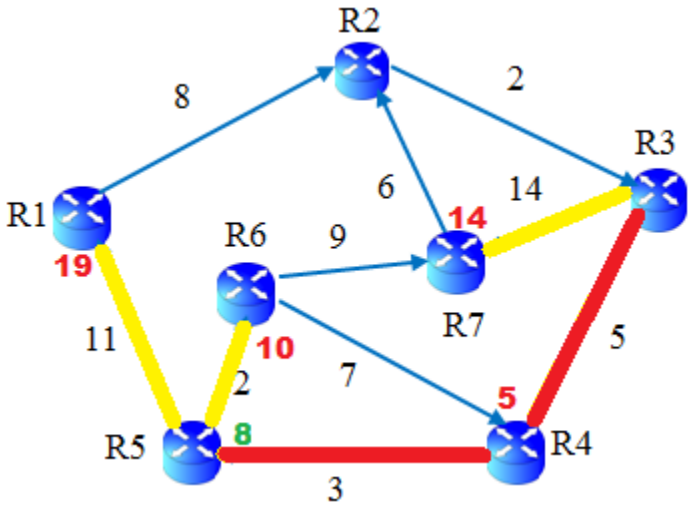


Figure 2.10 : graphe 6

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	

Tableau 2.6: Table 6

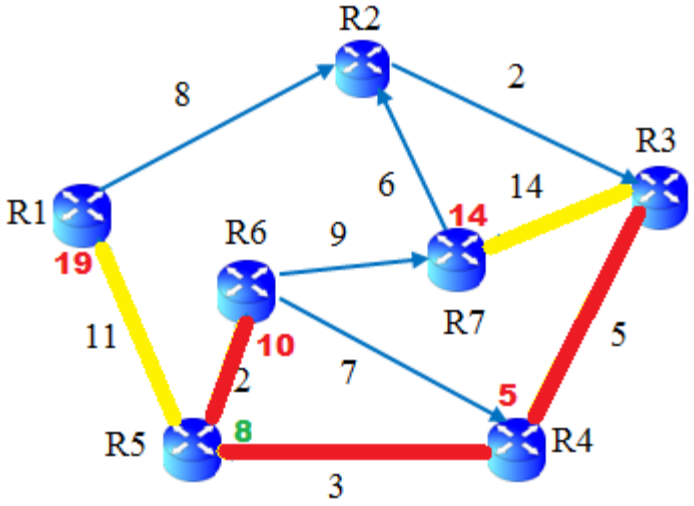


Figure 2.11: graphe 7

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	**R6**

Tableau 2.1: Table 7

Dans l'ensemble des sommets X – {R3, R4, R5, R6} on choisit un sommet de plus petite marque (ici R7, de marque y_{R7} = 19 et R1 de marque y_{R1} = 19) et on attribue à tous ses successeurs *k* la marque inf. {yB B}. On constate que cette opération ne change aucune marque précédente.

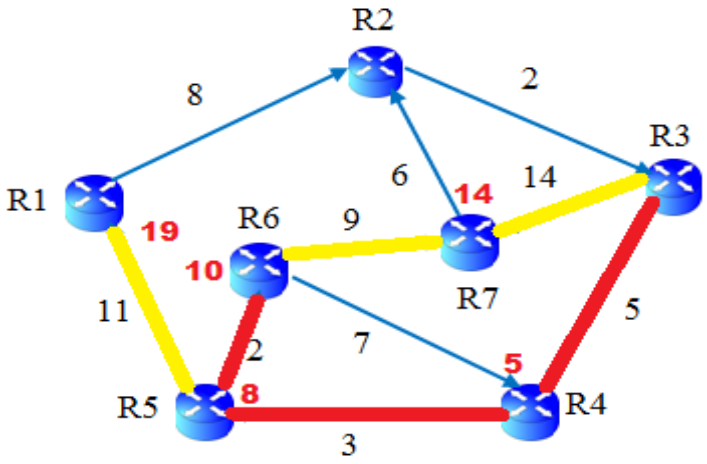


Figure 2. 12 : graphe 8

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	**R6**
	19_{R5}	∞				19_{R6}	

Tableau 2.1: Table 8

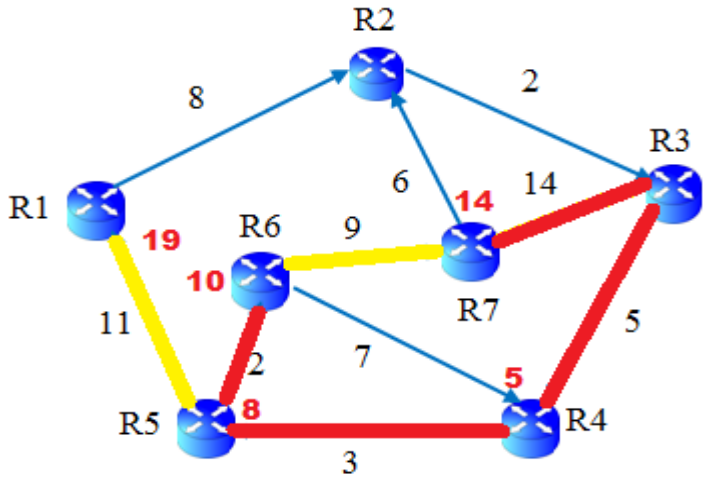


Figure 2. 13 : graphe 9

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	**R6**
	19_{R5}	∞				14_{R3}	**R7**
	19_{R5}	20_{R7}					

Tableau 2.9: Table 9

Dans l'ensemble des sommets X – {R3, R4, R5, R6, R7} on choisit le sommet de la plus petite marque (ici R1, de marque yR1 = 19), et on attribue à tous ses successeurs k la marque inf $\{y_k, y_{R1} + c_{R1}k\}$. On obtient donc :

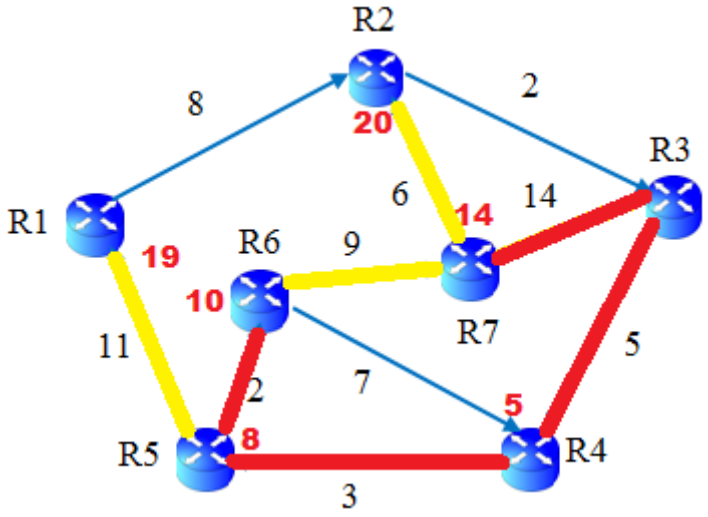


Figure 2. 14 : graphe 10

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	**R6**
	19_{R5}	∞				14_{R3}	**R7**
	19_{R5}	20_{R7}					**R1**

Tableau 2.10: Table10

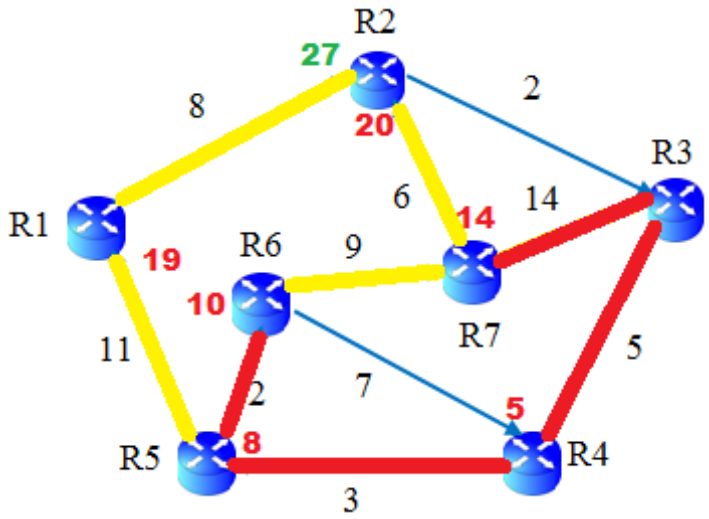


Figure 2. 15 : graphe 11

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	**R6**
	19_{R5}	∞				14_{R3}	**R7**
	19_{R5}	20_{R7}					**R1**
	27_{R1}	20_{R7}					**R2**

Tableau 2.11: Table 11

Enfin dans l'ensemble des sommets X – {R3, R4, R5, R6, R7, R1} on choisit le dernier sommet de plus petite marque (ici R2, de marque $y_{R2} = 20$). Ce sommet n'a pas de successeur. On constate donc également que cette opération ne change aucune marque comme la montre la figure suivante :

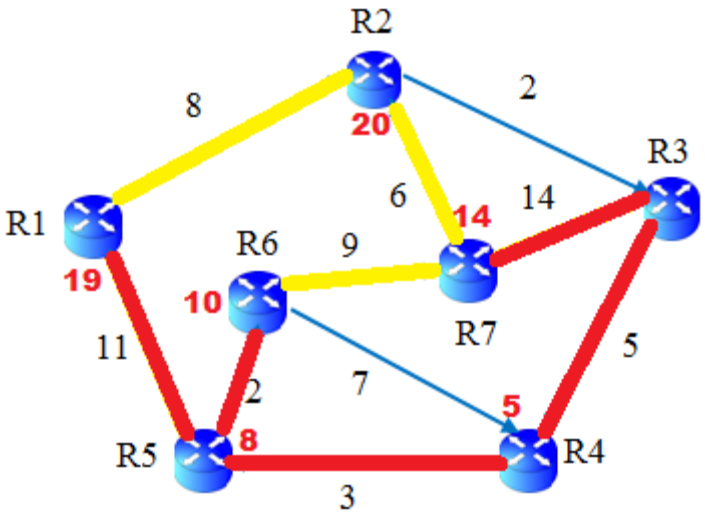


Figure 2. 16 : graphe 12

R3	R1	R2	R4	R5	R6	R7	On garde
0	∞	∞	∞	∞	∞	∞	**R3**
	∞	∞	5_{R3}	∞	∞	14_{R3}	**R4**
	∞	∞		8_{R4}	∞	14_{R3}	**R5**
	19_{R5}	∞			10_{R5}	14_{R3}	**R6**
	19_{R5}	∞				14_{R3}	**R7**
	19_{R5}	20_{R7}					**R1**
		20_{R7}					**R2**

Tableau 2.12: Table12

Tous les sommets du graphe ayant été choisis, l'algorithme est terminé et on obtient les longueurs des plus courts chemins de ‘‘a ’’ aux autres sommets (voir figure 2.16).

Chemins

Les chemins les plus courts par rapport à ''R3'' sont :

R3 : R3 ; 0 **R1 : R1 →R5 →R4 →R3 ; 19**

R4 : R4 →R3 ; 5 **R2 : R2 →R7→R3 ; 20**

R5 : R5→R4 → R3 ; 8 **R6 : R6 →R5 →R4 → R3 ; 10**

R7 : R7 →R3 ; 14

- La longueur du plus court chemin de R3à R1 est alors dist (19)
- La chaîne de poids minimum se lit « à l'envers », de R3 à chacun de ses prédécesseurs successifs.

1.5. Conclusion

Dans ce chapitre, nous avons présenté la théorie des graphes, le routage, et nous avons expliqué l'algorithme de Dijkstra par un exemple bien détaillé. L'algorithme de Dijkstra est utilisé dans des différents domaines. Il trouve son utilité dans les itinéraires routiers où le coût est la distance pour calculer les meilleurs chemins à travers un réseau si on cherche le trajet le plus court ou le temps pour le trajet le plus rapide. On peut utiliser n'importe quel attribut de coût de réseau valide lors de la définition du meilleur itinéraire. Il est utilisé aussi dans les protocoles de routage interne « à état de liens », tels que le protocole OSPF ou IS-IS, qui permettent un routage internet très efficace des informations en cherchant le parcours le plus efficace.

Chapitre 3

LES CIRCUITS LOGIQUES PROGRAMMABLES

3.1 Introduction

Il y a quelques années, la réalisation d'un montage en électronique numérique impliquait l'utilisation d'un nombre important de circuits intégrés logiques. Ceci avait pour conséquences un prix de revient élevé, une mise en œuvre complexe et un circuit imprimé de taille importante [48].

Le développement des mémoires utilisées en informatique fut à l'origine des premiers circuits logiques programmables (FPGA). (Field-Programmable Gate Array, réseau de portes programmables). Ce type de produit peut intégrer dans un seul circuit plus d'un million de portes logiques programmables par l'utilisateur. Sa mise en œuvre se fait très facilement à l'aide des méthodes de conception assistée par ordinateur (CAO), donc le rôle des FPGA est d'intégrer des circuits logiques complexes. Ces circuits sont susceptibles d'être reconfigurés (architecture programmée modifiable) partiellement ou entièrement suivant l'application [49].

A cet effet, ce chapitre s'inscrit dans un contexte qui traite les circuits logiques programmables et leurs positions au sein des autres systèmes digitaux. Nous allons ensuite étudier la carte FPGA utilisée dans cette thèse et décrire l'apport de cette technologie programmable sur les réseaux de télécommunications.

3.2 L'évolution des coûts des circuits numériques

Dans les années précédentes, des statistiques ont été faites sur les circuits numériques, elles reposent sur les trois aspects des circuits logiques qui sont : le combinatoire, le séquentiel et l'hybride et ont trouvé qu'au bout de 10 ans, le prix d'une porte logique a été divisé par 200 d'où l'intérêt économique du numérique.

A cet effet, le prix d'une porte logique se réduit de 40% par an. Le diagramme suivant montre ces statistiques d'évolution des coûts :

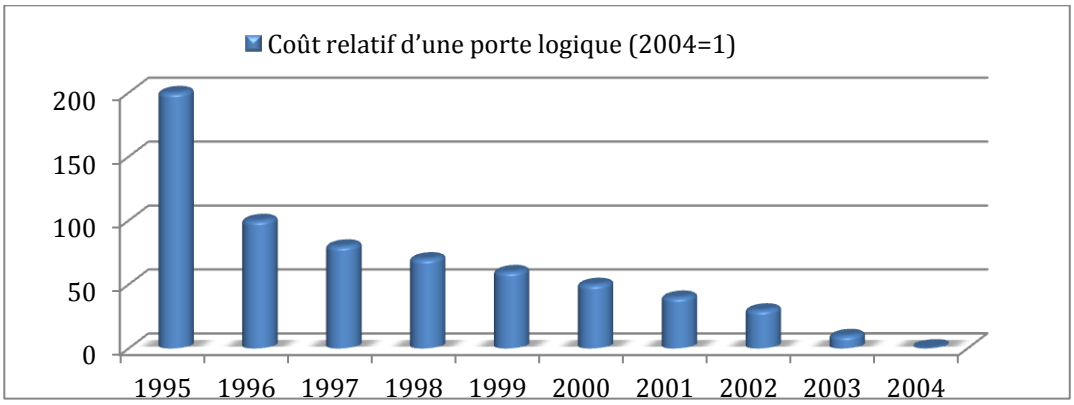


Figure 3. 1 : Diagramme d'évolution des coûts

Le tableau suivant montre que l'évolution des circuits numériques est le fruit d'un ensemble de facteurs :

Besoins croissants en circuits spécialisés	**Evolution rapide de la Technologie**	**Evolution des outils de conception "haut-niveau"**
- Produits de plus en plus complexes. - Contraintes:(performance, coût).	- Généralisation des «Systèmes sur Puce ». - Espace de conception trop grand.	- Produire une architecture à partir d'un algorithme. - Exploration automatique de l'espace de conception. - Outils d'estimations (performances, surface, etc.).

Tableau 3.1 : les facteurs d'évolution des circuits numériques.

3.3 Les technologies de Mémorisation

Voici d'une manière générale, l'arborescence des mémoires disponibles à nos jours [50] :

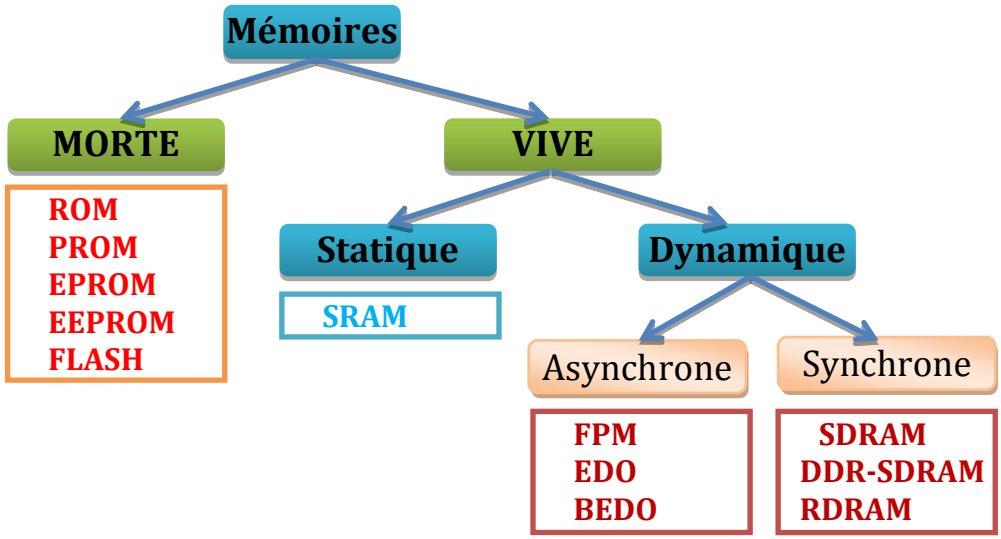


Figure 3. 2 : Les différents types de mémoires.

L'ensemble des caractéristiques de ces mémoires sont récapitulées comme suit :

- **Les ROM (R**ead **O**nly **M**emory**):** Mémoires figées par le concepteur à lecture seule et non modifiables.
- **Les PROM (P**rogrammable **R**ead **O**nly **M**emory**):** Mémoires programmables une fois par l'utilisateur avec un équipement spécialisé (tableau de fusibles).

- **Les EP-ROM (E**rasable **P**rogrammable **R**ead **O**nly **M**emory**):** Mémoires programmables électriquement et effacement par des rayons ultra-violets au bout d'un certain temps (Quelques minutes).
- **Les EEP-ROM (E**lectrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory**):** Mémoires programmables électriquement à lecture seule, effaçables électriquement (Quelques milli-secondes).
- **Les mémoires FLASH:** Elles sont une version plus évolué des EEP-ROM avec avantage d'être plus facile à programmer et à effacer.
- **Les S-RAM (S**tatic **R**andom **M**émory**):** Mémoires volatiles avec cellule de base à plusieurs transistors (accès rapide, consommation plus, coûteux). La volatilité correspond au non disponibilité de l'information lorsqu'il n'y a pas d'alimentation.
- **Les RAM dynamiques:** Mémoires volatiles qui nécessitent des rafraichissements périodiques de l'information afin de la conserver avec une cellule de base à un transistor (densité forte, accès lent).

3.4 Les circuits logiques Programmables

Le principe de la logique programmable remonte au début des années 1960, le concept ayant été proposé par « G. Estrin ». Il a fallu attendre les années 1980 pour que les premières réalisations matérielles apparaissent sur le marché. L'apparition de ce type de circuit s'est d'abord faite au travers de circuits logiques programmables simples de type PAL (Programmable Array Logic), qui se programment comme des mémoires non volatiles de type ROM et sont utilisés pour implémenter des fonctions combinatoires simples, telles des décodeurs d'adresse, ou des contrôleurs de bus [51].

Avec les évolutions en micro-électronique, différentes familles de circuits programmables ont commencé à apparaître : Les CPLD (Complex Logic Programmable Device), puis les FPGA (Field Programmable Gate Arrays), introduits par la société Xilinx en 1985.

L'industrialisation de ce type de circuits s'est faite à grande échelle avec l'apparition de circuits de plus en plus performants et reprogrammables à volonté. Les circuits de type FPGA les plus récents offrent désormais l'équivalent de dix millions de portes logiques programmables, à des fréquences de fonctionnement atteignant les 200MHz.

D'une manière générale, il existe deux alternatives ou solutions qui sont:

✓ *Une solution logicielle* **:** Elle est nommée aussi solution programmable de type

« Processeur » où un traitement séquentiel relativement lent et une programmation dépendante du composants (DSP, Microprocesseur et Microcontrôleur…).

✓ *Une solution matérielle*: Elle est nommée aussi solution programmable de type «Logique» où un traitement parallèle en temps réel et une programmation architecturale avec un langage de description matériel HDL (Méthodologie de CAO) indépendante du composant (ASIC et FPGA…).

Les caractéristiques de ces circuits sont :

a. *Les circuits du type DSP/Microprocesseurs* : Un rapport performance/coût faible, un temps de conception très court et une grande souplesse d'utilisation.

b. *Les circuits du type spécialisé ASIC* : Très performants mais avec un cycle de conception long et une architecture figée.

c. *Les circuits du type FPGA :* Des performances proches des ASIC**,** un coût unitaire intermédiaire et un cycle de conception moyen et une architecture modifiable. Voici un tableau récapitulatif de comparaison des différentes solutions numériques :

	ASIC	**FPGA**	**DSP**
Performance	Très élevées	élevées	faibles
Taille	Faibles	moyenne	élevées
Consommation	Faibles	moyenne	Très élevées
Intégration	Système sur puce	Système sur puce	Composants supplémentaires
souplesse	Fonctions figées	reconfigurable	Programmable
Mise en œuvre	Complexe	Complexité moyenne	Complexité moyenne
Coût du composant	Très élevés	Moyen	faibles

Tableau 3. 2 : Comparaison des différentes solutions numériques

Le tableau 3.2 présente les avantages et inconvénients respectifs des architectures les plus couramment utilisées pour le traitement numérique du signal. Les circuits FPGA apparaissent comme un bon compromis, tant du point de vue de leur prix de revient limité que de leurs performances, leur permettant d'être intégrés dans les architectures de traitement du signal à haut-débit.

Les fréquences de fonctionnement du mode séquentiel dépassent aujourd'hui 2 GHz, la réduction du temps de cycle ne suffira pas à compenser les insuffisances de ce mode de fonctionnement. La fréquence d'horloge ou de fonctionnement des FPGA est relativement

faible devant les microprocesseurs et les DSP, elle ne dépasse pas quelques centaines de Mégahertz, mais cette faiblesse est largement compensée et même surpassée grâce au parallélisme de traitement. L'exploitation du parallélisme est une technique en pleine expansion dans les circuits numériques FPGA qui sont une alternative des DSP.

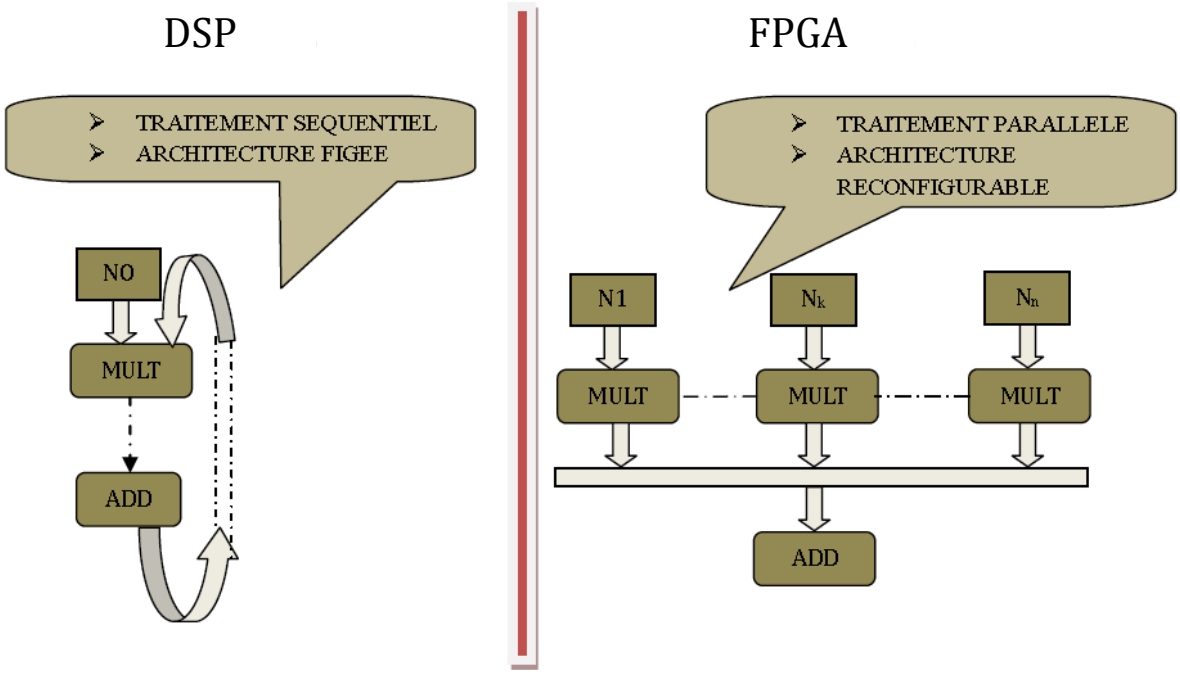


Figure 3.3 : Schéma comparatif d'un DSP et d'un FPGA [52]

Les circuits séquentiels ont une architecture matérielle figée et inversement pour les circuits logiques reconfigurables FPGA, on fait une adaptation de l'architecture du composant en fonction des algorithmes.

3.5 Les différents types des circuits Logiques Programmables

Auparavant, la distinction était nette entre le logiciel et le matériel, le circuit logique programmable FPGA est venu s'introduire comme un hybride entre les deux approches. Les circuits logiques programmables et reprogrammables architecturalement sont classifiés en trois grandes familles : les PLD, CPLD et les FPGA. La figure suivante illustre les différents types selon la technologie utilisée [52].

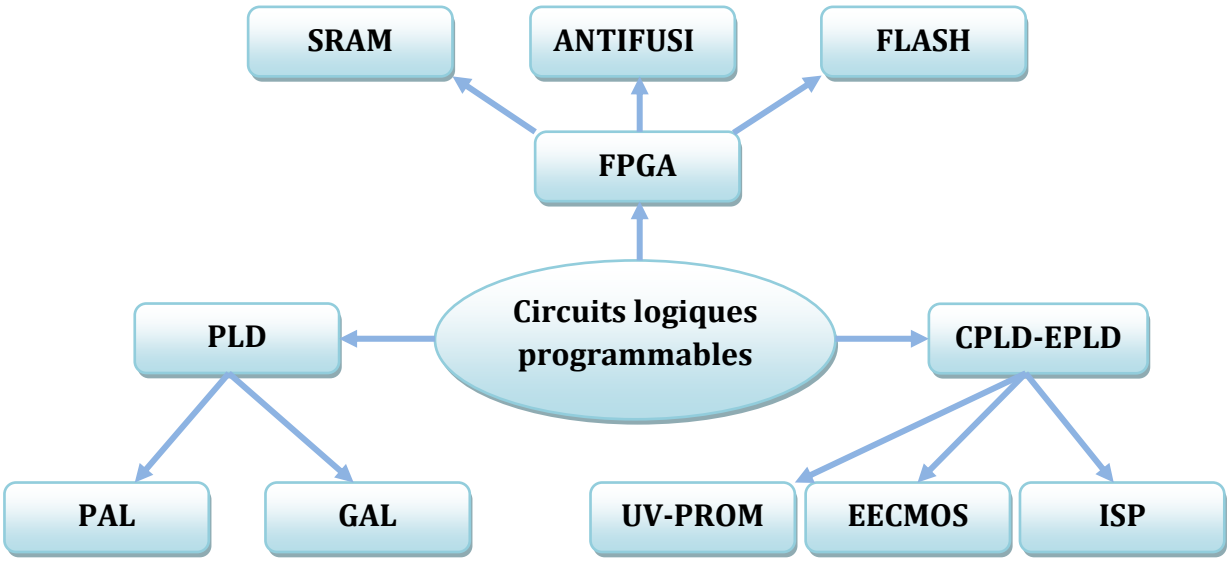


Figure 3. 4 : Diagramme des différents types de circuits logiques programmables

- *Les PLD (Programmable Logic Device)* : Famille des circuits programmables qui comprend les PAL, GAL.
- *PAL (Programmable Array Logic)* : Ce sont les circuits logiques programmables les plus anciens. Les PAL sont programmés par destruction de fusibles. Ils ne sont donc programmables qu'une fois.
- *GAL (Generic Array Logic)* : Les GAL sont des PAL effaçables électriquement, qui utilisent la technologie CMOS.
- *Les CPLD ou EPLD (Erasable Programmable Logic Device)* : Ces circuits ont une capacité en nombre de portes et en possibilités de configuration qui sont supérieures à celle des GAL.
- *ISP (In System Programmable)* : Circuit que l'on peut programmer même lorsqu'il est en place sur l'application.
- *Les FPGA (Field Programmable Gate Array)* : Ces circuits sont une évolution des CPLD. Récemment, ils intègrent également des mémoires entières, des multiplieurs et même des noyaux de processeur.

3.6 Les FPGAs

3.6.1 Critères de choix du circuit programmable FPGA

Les FPGA sont développés récemment grâce aux progrès de la technologie VLSI [53], l'apparition de ce type de circuits est une révolution des systèmes digitaux et ouvrant des perspectives de traitement numérique inaccessibles auparavant. La fin des années 80 a vu l'apparition des premiers circuits FPGA qui sont des circuits intégrés que l'on peut configurer en un temps relativement court pour réaliser n'importe quelle fonction logique « câblée » à bas

coût par une programmation de ses cellules logiques et ses interconnexions avec une restriction de ne pas épuiser les ressources du FPGA [54]. Typiquement, un circuit FPGA haute densité peut contenir jusqu'à plusieurs millions d'éléments programmables. Pour réussir une application à base de FPGA et afin d'obtenir un système plus performant, consommant un minimum de puissance, il est nécessaire de respecter un certain nombre de règles comme :

- Bien connaître les caractéristiques du FPGA ciblé pour assurer son adéquation avec les besoins du projet.
- Elaborer une méthodologie de conception.
- Maîtriser les outils d'implémentation et de choisir des outils de synthèse de qualité.

La conception sur les circuits FPGA est un challenge dans lequel l'objectif est de trouver le bon compromis entre densité, flexibilité et performances temporelles.

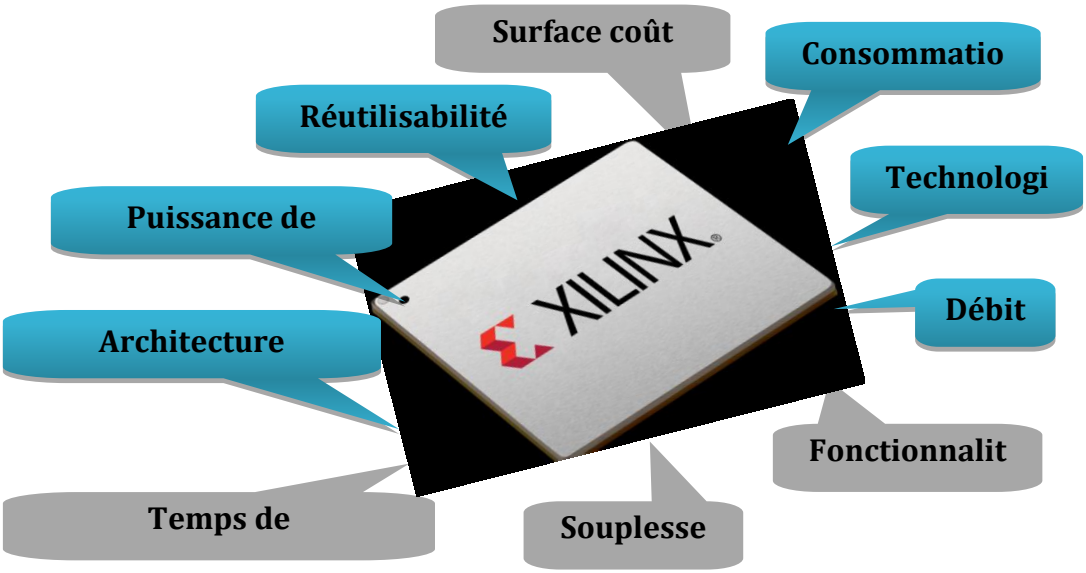


Figure 3. 5 : Critères de choix du circuit logique programmable FPGA

3.6.2 Différents domaines d'application des FPGA

Les FPGA ont fait révolutionner certains domaines de contrôle numérique et sont de plus en plus utilisés pour intégrer des architectures numériques complexes. Ils sont devenus les plus populaires en matière d'implantation et de prototypage des circuits numériques après leur apparition sur le marché en 1984**.** Leurs réussites sont dues à l'aspect de leur programmation. Leurs utilisations actuelles couvrent les deux domaines : civil et militaire. Parmi ces applications nous citons :

1- Informatique : Périphériques spécialisés.

2- Machinerie industrielle : Contrôleur pour machines.

3- Télécommunications : Traitement d'images, Filtrage, Routage.

4- Instrumentation : Équipement médical, Prototypage.

5- Transport : Contrôle d'avions et métros.

6- Aérospatiale: Satellites, Radar, Communication protégée, la détection ou la surveillance.

etc...

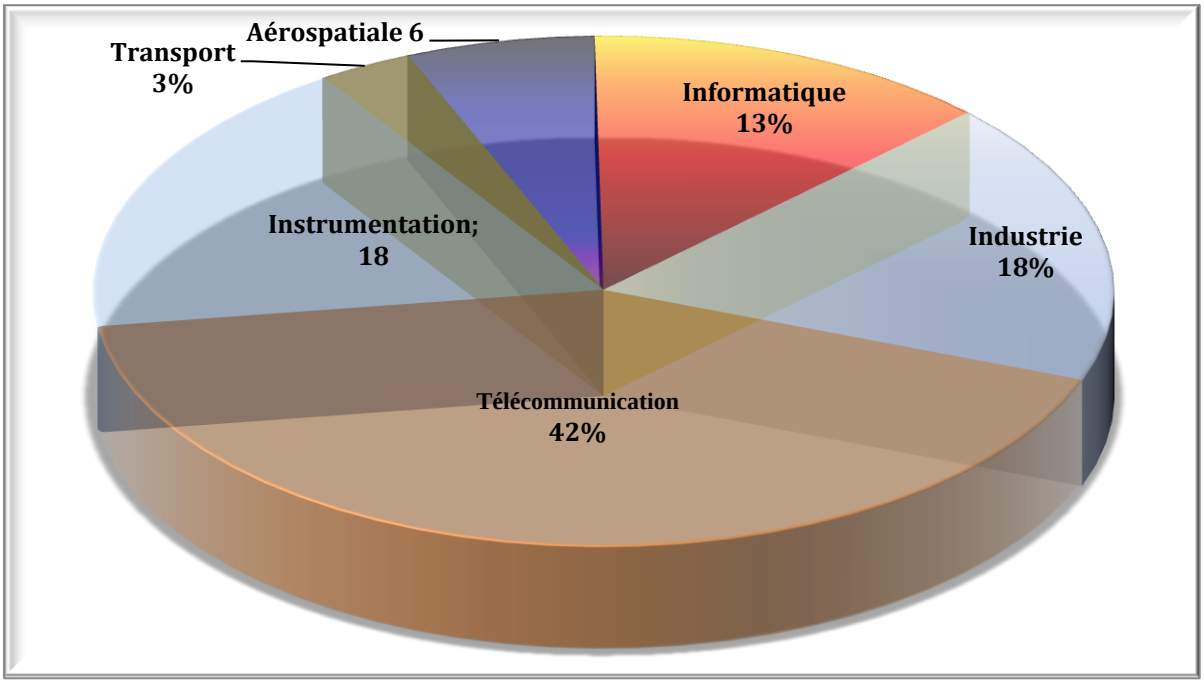


Figure 3. 6 : Différents domaines d'application des FPGA

3.6.3 Les fabricants de FPGA

L'ensemble des firmes (Principaux fondeurs) qui conçoivent ce type de circuits sont : Actel, Altera, Atmel, Cypress, Lattice, Minc, QuicLogic, Xilinx et d'autres.

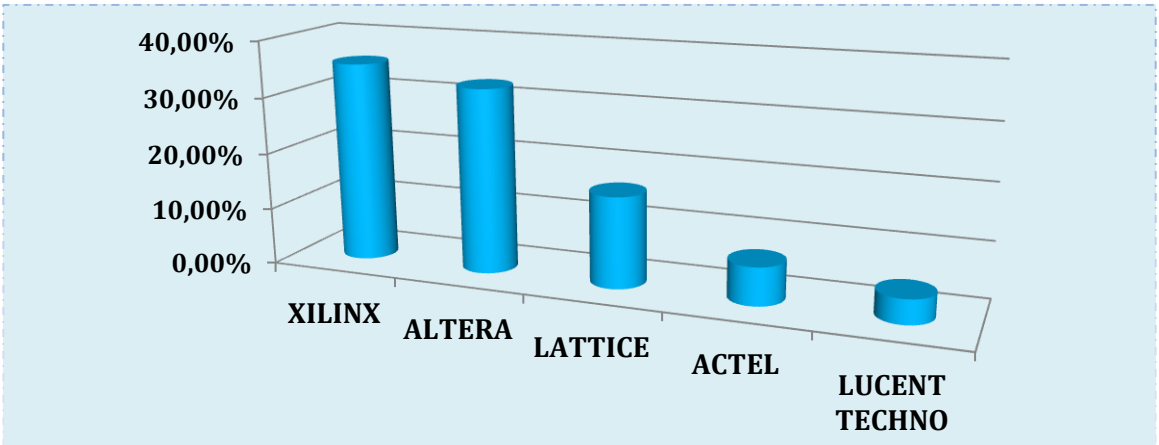


Figure 3.7 : Statistiques du marché occupé par les vendeurs d'FPGA [55].

3.6.4 Configuration et reconfiguration des FPGA

Un système reconfigurable est un système qui est constitué de composants ou entités à architecture modifiable afin de répondre à un objectif bien déterminé. Ce système reconfigurable dispose d'un mécanisme permettant de choisir une nouvelle configuration et de la mettre en place dans le cadre du processus en question [56].

Les circuits FPGA sont un type de ces circuits reconfigurables. Ils sont programmables ou configurables sur les cartes sur quelles ils sont implantés par l'utilisateur. Cette reconfigurabilité est une propriété nécessaire face aux systèmes à charges et contraintes variables. La reprogrammation du FPGA est décrite dans la figure suivante.

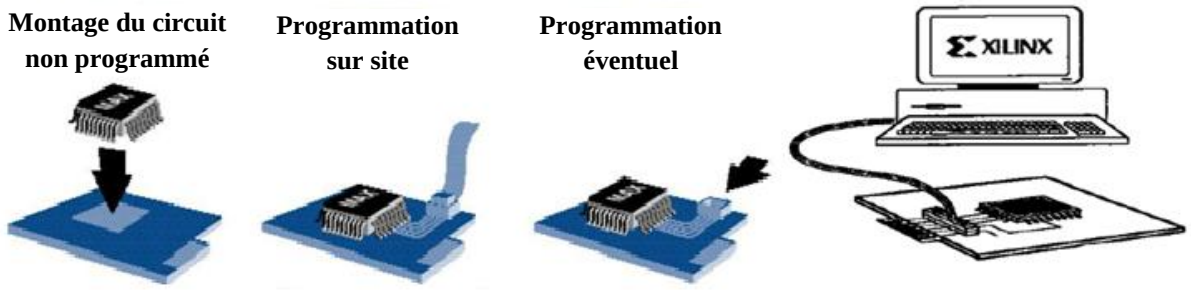


Figure 3.8 : Reprogrammation sur site d'un FPGA

3.6.5 Technologies de programmation des FPGA

Pour franchir les inconvénients sus-mentionnés des mémoires, et dans le but de faire un ensemble de technologies complémentaires adaptables suivant l'environnement des cahiers de charges, il existe trois types de FPGAs reprogrammables suivant la technologie de mémorisation pour répondre aux différentes applications [57].

Les trois principales technologies du FPGA sont :

- **T**echnologie de programmation par **RAM**.
- **T**echnologie de programmation par **EEPROM** ou **FLASH**.
- **T**echnologie de programmation par **ANTI-FUSIBLE.**

3.6.5.1 Technologie à base de RAM (XILINX et ALTERA)

Cette technologie permet d'avoir une reconfiguration rapide des FPGA. Les points de connexions sont des ensembles de transistors commandés [58]. L'inconvénient majeur de cette technologie c'est qu'elle nécessite beaucoup de places et il est nécessaire de sauvegarder le design du FPGA dans une autre mémoire Flash.

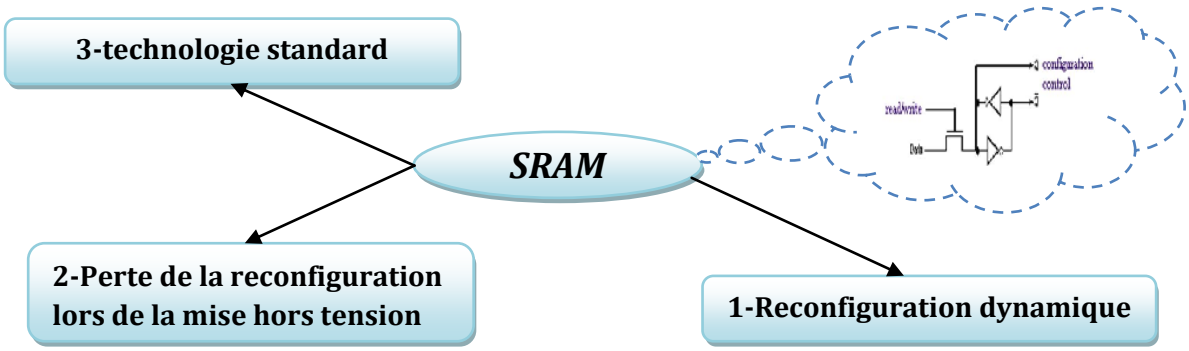


Figure 3.9 : Caractéristiques des technologies SRAM

3.6.5.2 Technologie à base d'EEPROM ou FLASH (LATTICE et ACTEL)

Cette technologie garde sa configuration mais un nombre limité de configuration avec une configuration plus lente par rapport à SRAM.

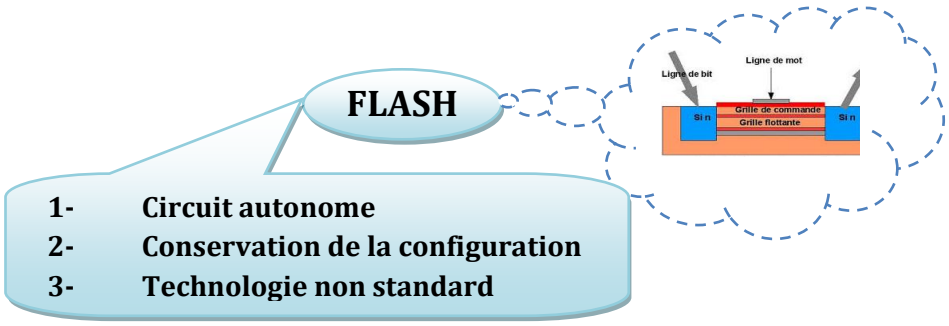


Figure 3.10 : Caractéristiques des technologies FLASH

3.6.5.3 Technologie à base d'anti-fusibles (ACTEL)

Les points de connexions sont du type ROM, c'est-à-dire que la modification du point est irréversible. Pour comprendre le mécanisme de connexion sans rentrer dans les détails des semi-conducteurs, on considère que le point de connexion est le point de rencontre de deux segments conducteurs ou lignes conductrices. Le nom anti-fusible vient du fait que l'état initial du fusible ou la couche isolante est présente et il n'y a pas de contact pour l'établir, il faut détruire le fusible ce qui est contradictoire au fonctionnement habituel d'un fusible. Des composants moins génériques mais plus petits et plus rapides ont été développés.

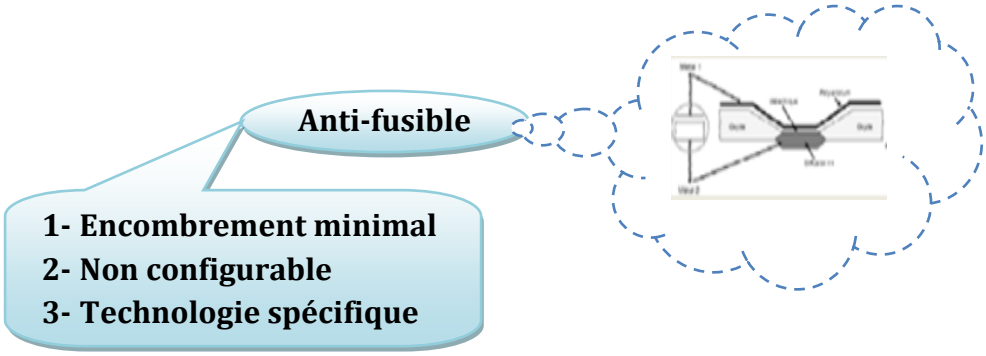


Figure 3.11 : Caractéristiques des technologies Anti-fusible

3.7 Architecture interne des FPGA

On appelle FPGA, quelques fois LCA, abréviation anglaise de « **L**ogic **C**ell **A**rray » signifiant réseau de cellules logiques. Pour réussir à implanter un système dans un FPGA de manière efficace, il est indispensable de bien connaître sa structure interne et ses limites du point de vue performances. Les composants logiques programmables sont des circuits composés de nombreuses cellules logiques élémentaires librement assemblables. Celles-ci sont connectées de manière définitive ou réversible par programmation afin de réaliser les ou les fonctions numériques désirées. Un FPGA est un circuit intégré avec une structure adaptable par l'utilisateur et composée d'un réseau de cellules élémentaires ou d'éléments logiques programmables CLB et IOB répartis régulièrement et reliés entre eux grâce à des connexions qui forment une matrice de routage programmable pour obtenir un comportement spécialisé du circuit dans sa globalité, puisque tous les éléments sont programmables [59].

L'ensemble des systèmes reconfigurables FPGA est subdivisé en trois catégories suivant les fonctions préexistantes et des possibilités de les interconnecter. Ces catégories sont : des systèmes reconfigurables nommés "**grain fin**", "**grain moyen**" et "**grain large**".

L'architecture interne des FPGA est différente d'un fondeur à un autre et même entre les différentes gammes du même constructeur mais rien n'empêche que leurs ressemblances peuvent être rassemblées dans le schéma représentatif de la figure suivante :

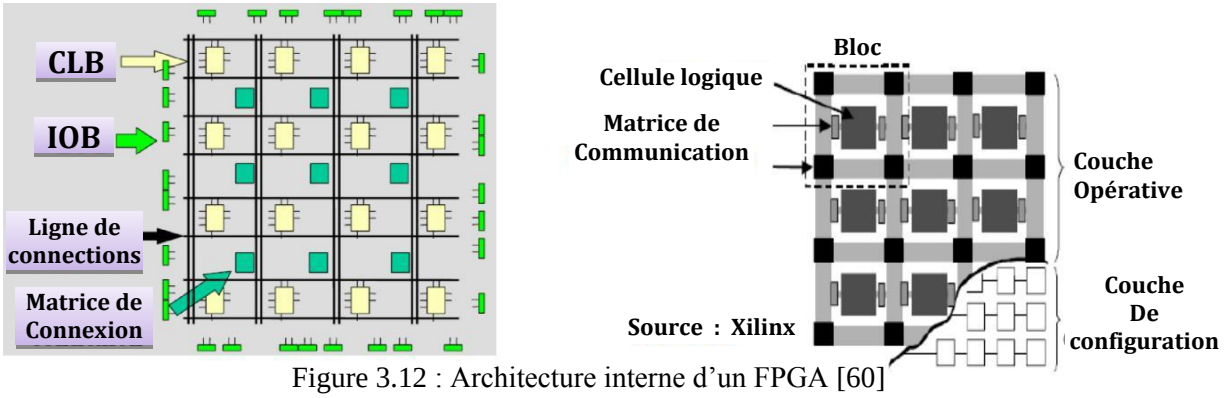


Figure 3.12 : Architecture interne d'un FPGA [60]

❖ Les macro-cellules internes sont appelées :

- **CLB** qui est la dénomination adoptée par Xilinx et abréviation anglaise de « **C**onfigurable **L**ogic **B**lock », signifiant bloc logique configurable.
- **LC** qui est le nom choisi par Cyprès et abréviation anglaise de « **L**ogic **C**ell », signifiant cellule logique.
- **LE** qui est l'appellation d'Altéra abréviation anglaise de « **L**ogic **E**lement» signifiant élément logique.

❖ Les macro-cellules sur la périphérie sont appelées : **IOB** abréviation anglaise de « **I**nput **O**utput **B**lock », signifiant bloc logique d'entrées sorties.

❖ L'ensemble des points de connexion est appelé **PIP**, abréviation anglaise de «**P**rogramme **I**nterconnect **P**oints». La granularité des FPGA par les macro-cellules CLB nous permet d'implémenter des fonctions logiques « combinatoires ou séquentielles » complexes car chaque CLB est constitué d'une partie combinatoire et d'une partie séquentielle. Chaque fonction est décomposée en petites fonctions booliennes qui peuvent être contenues par de petites cellules élémentaires SLICES [61].
Ces dernières comportent des LUT pour la partie combinatoire et une ou des bascules (généralement de type D) pour la partie séquentielle.

Les architectures existantes peuvent être regroupées en trois grandes catégories suivant la manière dont les blocs logiques sont organisés comme le montre la figure suivante:

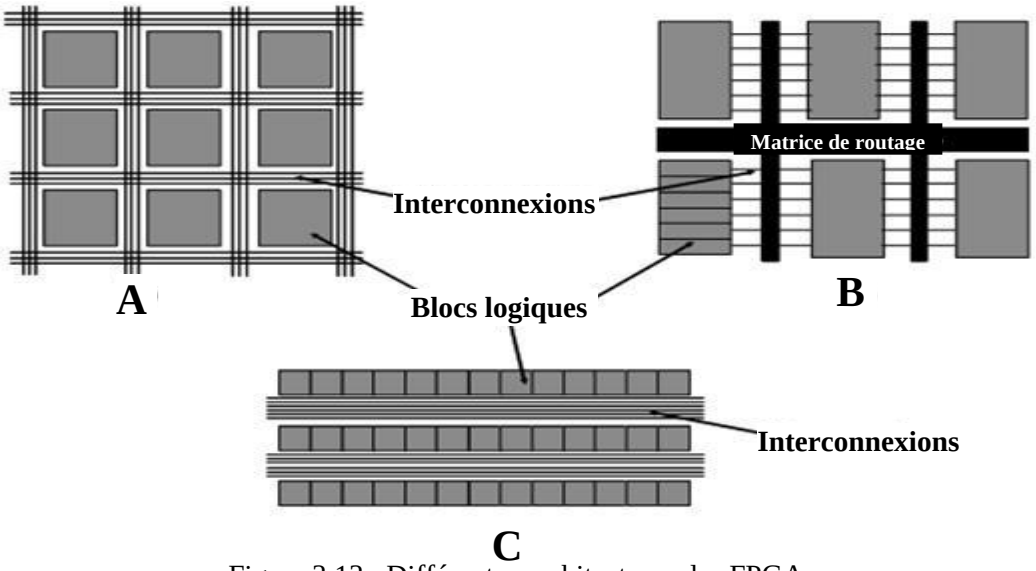


Figure 3.13 : Différentes architectures des FPGA

3.7.1 Architecture MULTI-COMPOSANTS

Dans un environnement multi-composant, plusieurs structures sont possibles pour un certain nombre de FPGAs qui admettent l'association [62], on cite comme exemple:

- Association de plusieurs FPGA

L'association de plusieurs FPGA prend différentes architectures suivant le besoin, on note :

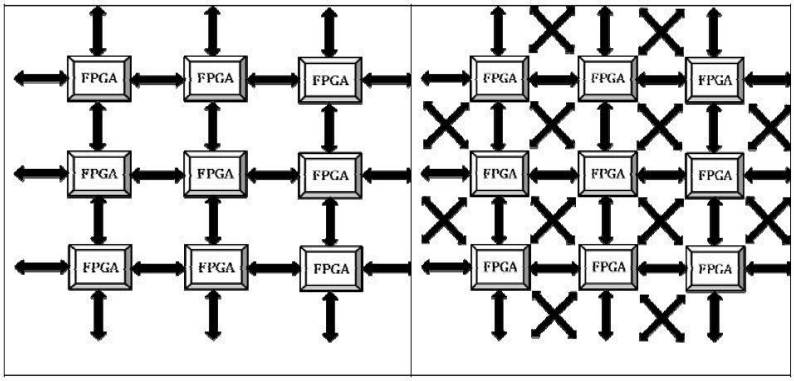


Figure 3.14 : Exemples d'association entre FPGA

- Association d'un microprocesseur et d'un FPGA

La combinaison entre un **FPGA** et un processeur est possible dans certains cas comme accélérateurs matériels où le système reconfigurable est directement couplé à un processeur, ce qui constitue un SOC**.** Le microcontrôleur ou le microprocesseur est généralement le hôte qui organise, initialise, charge les programmes…

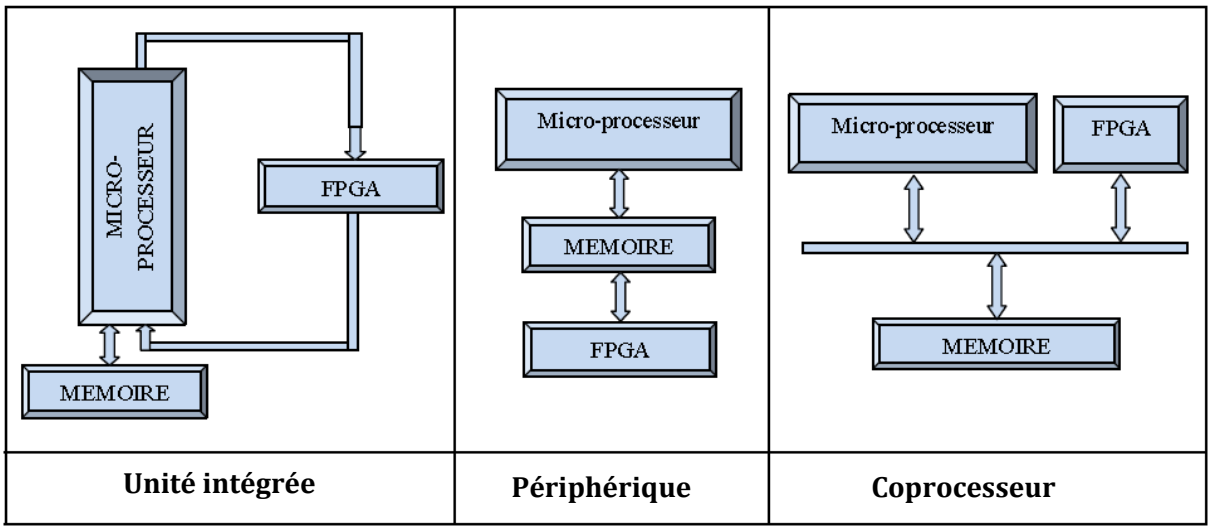


Figure 3.15 : Exemples d'association entre FPGA et Microprocesseur.

3.7.2 Avantages et inconvénients des FPGA

Les avantages et les inconvénients des FPGA sont multiples, on trouve :

Avantages	Inconvénients
Technologie « facile » à maîtriser. **T**emps de développement réduit. **R**eprogrammable. **I**déal pour le prototypage. **P**arallélisme de traitement. **F**lexibilité et possibilité de réduire **F**ortement concernant les délais de développement et de commercialisation. **L**a reconfiguration, parfois en temps réel.	**P**erformances non optimisées. **T**emps de réponse long par rapport aux **ASIC**. **P**rix unitaire peu élevé pour les très grandes séries.

Tableau 3.3 : Avantages et inconvénients des FPGAs [63].

3.7.3 Les deux grandes familles architecturales de FPGA

Les familles des FPGA peuvent se regrouper en deux groupes :

✓ Les circuits FPGA à base de « LUT » (Look Up Tables)

Les LUT ressemblent aux tables de vérité des fonctions logiques et réalisables par des mémoires de type SRAM qui permettent de mettre en œuvre le concept de prototypage (ou maquette) pour la vérification fonctionnelle des systèmes sur puce pour certaines applications. La fonction de la LUT est de stocker la table de vérité de la fonction combinatoire à implémenter comme le montre la figure suivante :

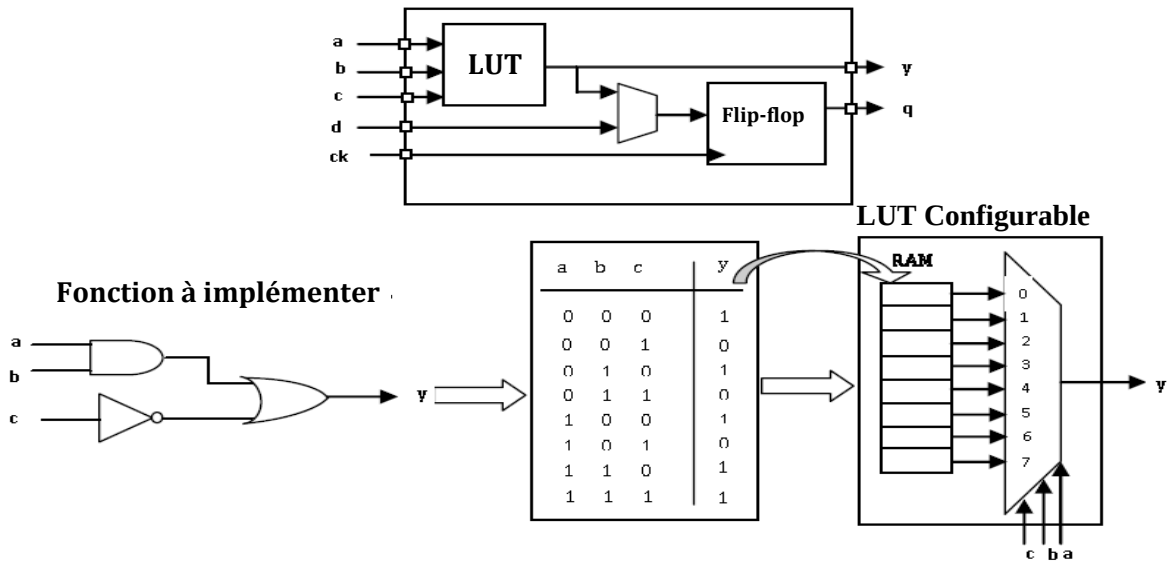
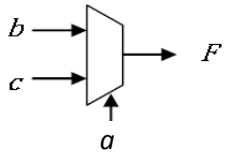


Figure 3.16 : Exemple d'implémentation sur LUT [64]

✓ Les circuits FPGA à base de multiplexeurs « MUX »

Les FPGA à base de multiplexeurs qui sont des microcellules à trois entrées capables de réaliser la fonction suivante :

Equation logique	Symbole
F=(a AND b) OR (ā AND b)	

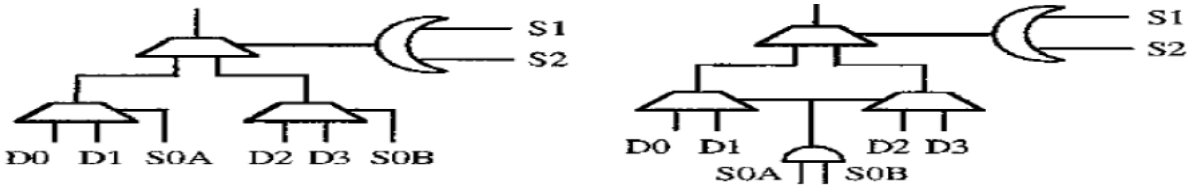


Figure 3.17 : Exemple d'implémentation sur des multiplexeurs.

3.8 Technologie du premier fondeur du FPGA « XILINX »

Le plus gros constructeur du marché est XILINX qui a introduit la série XC2000 (de 600 à 1500 portes) entre 1984 et 1985 avec des fréquences de fonctionnement pouvant atteindre les 420 MHZ. Depuis, d'autres séries sont apparues comme les XC3000, XC4000, XC5200, XC6200, ainsi que les XC9500 et la mise sur le marché du 1er FPGA XILINX en 1985. Puis l'apparition en 1992 du premier FPGA du constructeur ALTERA qui est le concurrent de XILINX, avec un type de circuits assez différent le FLEX 8000 (15000 portes max). Rapidement l'exploitation de la technologie EEPROM fut un an plus tard en 1993 puis le lancement du VIRTEX II /XILINX (jusqu'à 10 millions de portes) en 2001 et des FPGA de capacités supérieures à 50 millions de portes fonctionnant à des fréquences dépassant les *500 MHz* en *2005***.**

Le principe des FPGA de XILINX est de stocker la configuration dans une mémoire vive statique SRAM. Aujourd'hui des blocs de fonctionnalités supplémentaires dans quelques versions évoluées sont ajoutés et dédiés à des applications spécifiques. Ces fonctionnalités supplémentaires sont : Mémoire RAM, Petits multiplieurs, Blocs DSP, Cœurs de processeurs RISC et arbres de distribution d'horloge pour générer différents domaines d'horloge pour un bon synchronisme pour les FPGAs à grande capacité.

3.8.1 Structure des CLB

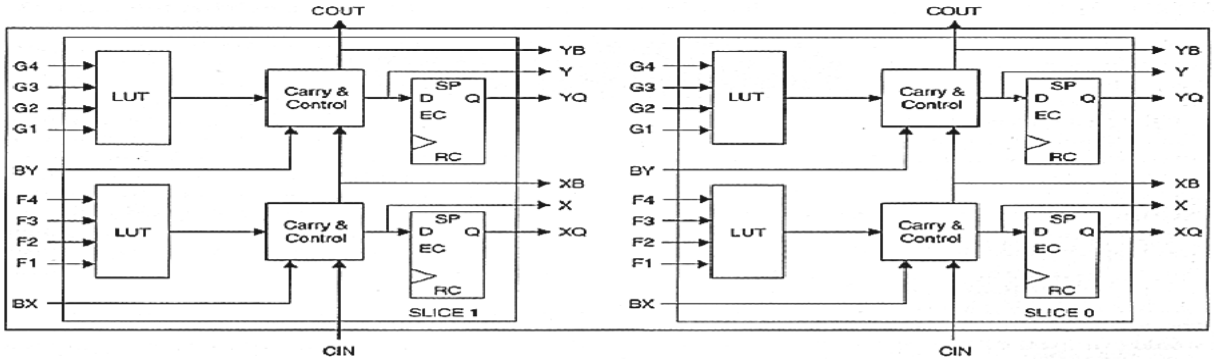


Figure 3.18 : Architecture d'un CLB de familles VIRTEX.

L'architecture simplifiée d'un slice est comme suit :

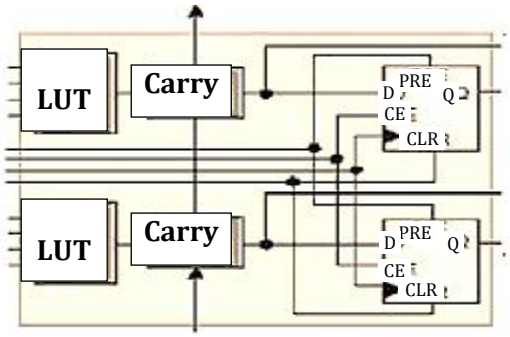


Figure 3.19: Architecture d'un SLICE de familles Virtex.

3.8.2 Structure des IOB

Les blocs d'entrées/sorties disposent de bascules aussi de contrôle à trois-états.

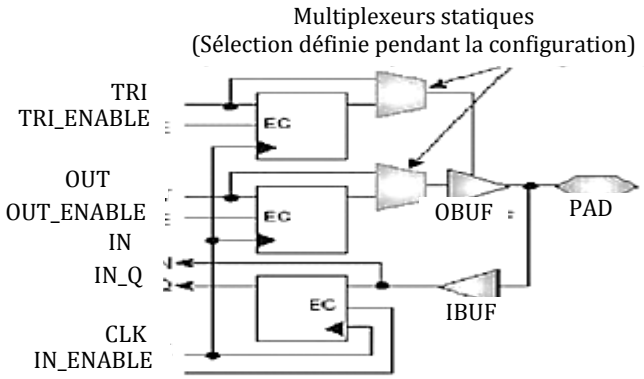
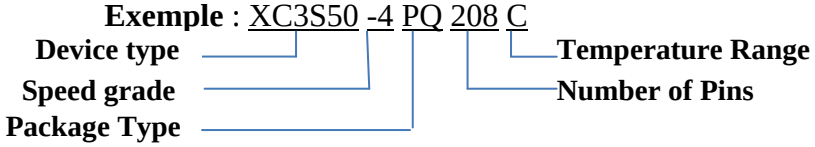


Figure 3.20 : Architecture d'un IOB de familles Virtex.

3.8.3. Nomenclature des Virtex :

Voici un exemple de référence de la famille VIRTEX et le détail de sa nomenclature [65] :



Device	**Speed grade**		**Package type/Number of pins**		**Temperature Range**	
XC3S50	**-4**	Standard performance	**VQ100 VQG100**	**100-pin (VQFP)**	C	Commercial (0^0 to 85^0c)
XC3S200	**-5**	High performance	**CP132 CPG 100**	**132-ball (CSP)**	I	Industrial(-40^0 to 100^0c)
XC3S1000			**TQ144 TQG144**	**144-pin (TQFP)**		
XC3S1500			**PQ208 PQG208**	**208-pin (PQFP)**		
XC3S2000			**FT256 FTG256**	**256-ball(FTBGA)**		
XC3S4000			**FG320 FGG320**	**320-ball (FBGA)**		
XC3S5000			**FG484 FGG484**	**484-ball (FBGA)**		

Figure 3.21 : exemple de référence de la famille VIRTEX

3.8.4. Evolution des FPGAs XILINX de la famille Virtex :

Famille	Année	Technologie	Max cellules	Détail techno
Virtex4	2004	90nm	200000LUT4	Triple oxyde (réduction de la consommation)
Virtex5	2007	65nm	330000LUT6	Lithographie par immersion (finesse de la gravure)
Virtex 6	2010	40nm	760000LUT6	Implants SIGe (augmentation de la vitesse)
Virtex 7	2012	28nm	2000000LUT6	High-k technology (réduit les courants de fuite)
En préparation : VIRTEX ULTRASCALE **avec technologie 3D 20nm**				

Tableau 3.4 : Evolution des FPGAs XILINX famille Virtex

3.9. Etude de la famille VIRTEX 7 de type (XC7V2000T):

La famille Virtex-7 de Xilinx est un circuit logique programmable de très haut de gamme en termes de densité, avec 2 millions de cellules logiques de base, et pour le nombre d'entrées/sorties disponibles pour l'utilisateur. Très flexible, elle est proposée comme alternative aux ASICs sous trois familles : LX, logique haute performance ; FX, traitement de signal très haute performance et SX, pour les traitements embarqués et connectivité rapide. Elle a été réalisée selon la technologie HPL 28nm [66].

Virtex-7 (XC7V2000T) est aussi un circuit logique programmable de grande capacité, il contient 6,8 milliards de transistors. Cette capacité est rendue possible par la technologie ‘‘Stacked Silicon Interconnect’’ (SSI). Cette technologie de Xilinx utilise une couche spéciale

de silicium appelé "silicon interposer" combiné avec Through-Silicon Vias (TSV). Le premier FPGA Xilinx utilisant cette technologie est Vertix-7 (XC7V2000T) [67] [68] qui intègre 4 puces FPGA dans le même package.

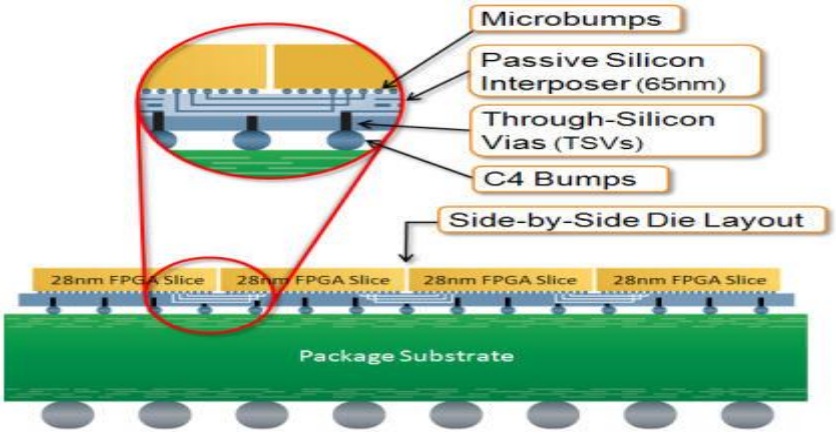


Figure 3.22 : L'architecture utilisé dans le Virtex 7-2000T [68].

La figure 3.22 représente l'architecture utilisée dans le Virtex 7-2000T proposée par *Xilinx* [68], où les puces FPGA sont implémentées au niveau du nœud de la technologie 28 nm, tandis que l'élément "Passive silicon interposer" est implémenté au niveau du nœud technologique 65 nm. L'implémentation du grand "silicon interposer" à ce nœud réduit le coût et augmente le rendement sans dégrader les performances. Le ‘‘silicon interposer’’ ajoute essentiellement quatre couches supplémentaires qui peuvent être utilisées pour connecter les FPGA les uns aux autres avec plus de 10.000 connexions entre chaque paire de matrice adjacente. Through-Silicon Vias (TSV) sont utilisés pour transmettre aux signaux de traverser ‘‘silicon interposer’’ aux Bumps C4 (perles de soudure) sur le fond de ‘‘silicon interposer’’ [66]. Ces perles de soudure sont ensuite utilisées pour connecter ‘‘silicon interposer’’ au ‘‘package Substrat ’’.

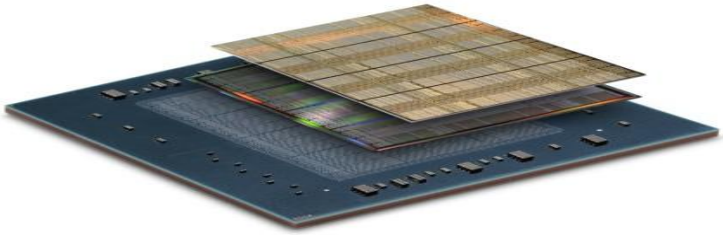


Figure 3.23 : Une vue de Virtex-7 2000T de Xilinx montrant le ‘‘package Substrat’’ (en bas), ‘‘silicon interposer’’ (au milieu), et quatre puces FPGAs (en haut) [68].

Le Virtex-7 2000T utilise la technologie ‘‘stacked silicon interconnect’’ pour intégrer deux FPGA ensemble sur la même carte qui donne par rapport à des connexions E/S standard plus de 100 fois la connectivité de bande passante par watt, le Virtex-7 2000T préserve le modèle

d'utilisation des FPGA traditionnelle. Les utilisateurs vont programmer l'appareil comme un grand FPGA avec le flux d'outil Xilinx et de la même méthodologie.

Avec le Virtex 7, la loi de Moore est dépassée, Xilinx pionnier de la technologie SSI pour obtenir des augmentations de capacité et de performances qui dépassent le rythme de la loi de Moore. Par conséquent, Virtex-7 FPGA offre plus de 3,5 fois la capacité de la génération précédente. Combiné avec la mémoire des familles de DSP et ressources E/S, les appareils Virtex-7 établissent des nouveaux critères de performance.

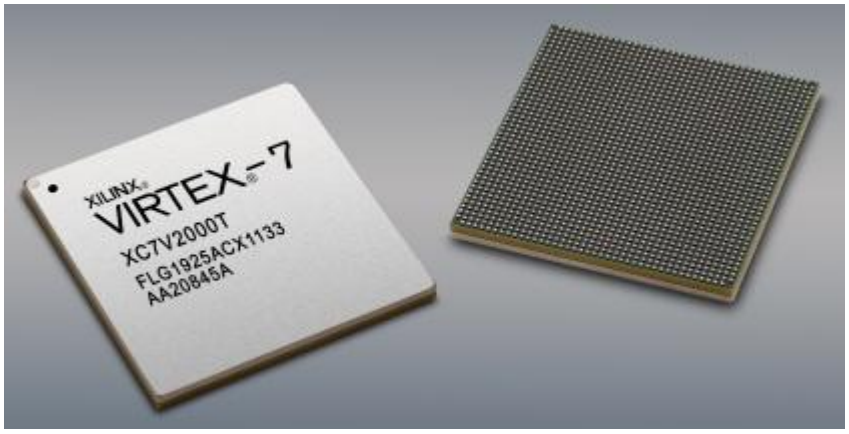


Figure 3.24 : Une vue de dessus et en bas de Virtex-7 dispositif de 2000T de Xilinx.

Pour avoir une faible consommation d'énergie du système, le virtex 7 élimine les interfaces d'E/S entre les différents circuits intégrés sur la même carte. Prenons l'exemple suivant fourni par Xilinx [69] qui compare un seul Virtex-7 2000T avec quatre des plus grands circuits intégrés monolithiques, comme illustré ci-dessous:

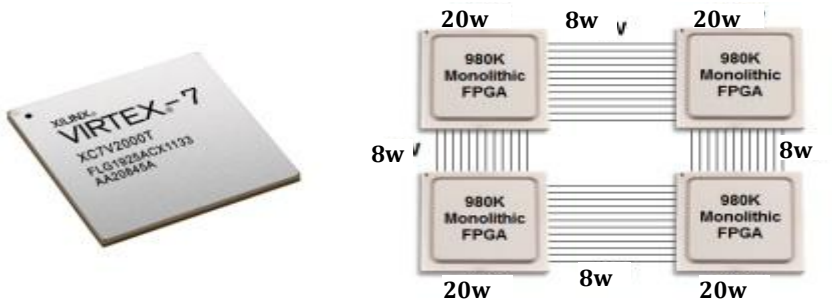


Figure 3.25 : Comparaison entre une seule Virtex-7 2000T et quatre circuits intégrés monolithiques

3.9.1. Présentation de la carte Virtex XC7V2000T

Le FPGA choisi dans ce travail est le XC7V2000TFLG1925 (famille: Virtex-7, dispositif: 2000T et package: FLG1925, donc de type FPGA: XC7V2000TFLG1925) qui est le plus grand FPGA de Xilinx [69].

FPGA type			Capacité logique				Capacité I/O			Prix ($)
Famille	**Devise**	**Package**	**LUT**	**REG**	**RAM**	**DSP**	**I/Os**	**LVDS**	**Bank**	
Virtex-5	LX330	FF1760	207360	207360	288	192	1200	600	32	2310
Spartan-6	LX150T	FG900	92152	184304	134	180	540	270	6	1050
Virtex-6	LX550T	FF1759	343680	687360	632	864	840	420	21	3850
Virtex-6	LX760	FF1760	474240	948480	720	864	1200	600	30	5320
Virtex-7	2000T	FLG1925	1221600	2443200	1292	2160	1200	576	24	14000

Tableau 3.5: capacité estimé des différents types FPGA

- ✓ Capacité logique (représentée par le nombre d'éléments logiques contenus dans le FPGA). Il existe quatre types d'éléments logiques qui sont pris en considération: Look-Up Table, registre (REG), Bloc RAM (RAM), et intégré DSP (DSP).
- ✓ Capacité I/O (représenté par le nombre d'E/S, le nombre de paires d'E/S en LVDS et le nombre de banques figurant dans le FPGA).
- ✓ Prix estimé (7 $ par ~ 1000 éléments logiques).

3.10 L'apport des FPGA à la télécommunication

Sous l'impulsion du progrès rapide et extraordinaire du numérique : l'amélioration de la qualité et des performances a toujours été une préoccupation constante des concepteurs de circuits numériques. C'est dans ce contexte que nous exploitons les FPGA pour faire une contribution à l'amélioration de la télécommunication en temps réel. Les contraintes majeures pour le routage de données dans les réseaux de télécommunication sont la satisfaction du compromis rapidité/précision d'une part et un taux de calcul élevé d'autre part. Une approche efficace pour résoudre ce genre de problème est aujourd'hui disponible et concrétisée par ces circuits logiques FPGA. L'introduction des FPGA est un remède à la complexité des algorithmes de routage ainsi qu'à la vitesse de traitement.

Le FPGA peut assurer toute la partie algorithmique de routage grâce à ses caractéristiques notamment le parallélisme de traitement. Parmi les caractéristiques de ce circuit, le routage dans les réseaux de télécommunication peut bénéficier de l'implantation des fonctions avancées irréalisables dans le domaine analogique, aucun impact des perturbations externes sur les algorithmes, la réalisation de systèmes sûrs et efficaces avec précision, la reprogrammabilité sur site sans changer de composant ni de câblage et un encombrement minimal où tous les algorithmes de routages et de contrôles sont intégrés sur une puce de quelques millimètres carrés. L'ensemble de ces caractéristiques est un acquis pour l'acheminement des données dans le réseau par une optimisation du rendement des convertisseurs statiques.

Finalement l'apport des FPGA à la télécommunication peut être résumé dans les points suivants :

- les FPGA sont des solutions numériques qui permettent d'approcher les avantages de l'analogique et de garder au même temps les avantages du numérique avec l'implantation d'algorithmes complexes, un temps de calcul réduit à quelques microsecondes.
- Eviter l'inconvénient majeur des solutions analogiques classiques qui réside dans l'influence des variations paramétriques engendrées par la sensibilité aux perturbations externes comme la chaleur.
- Pas d'entretien qui nécessite du temps et des pertes d'ordre économique à l'inverse des solutions analogiques.
- Implémentation de fonctionnalités supplémentaires qui ne sont pas réalisables en continu.
- Augmentation de la bande passante vis-à-vis des autres solutions numériques comme les DSP et microcontrôleurs ou microprocesseurs traditionnels.
- L'intégration sur une seule puce de plusieurs algorithmes de routage grâce à la configuration dynamique avec une grande flexibilité pour un changement de la topologie réseau.
- La possibilité de réduire fortement les délais de développement et de commercialisation.

L'utilisation des FPGA dans le routage de l'information dans un réseau ne nécessite pas d'espace, ce qui est équivalent à un encombrement minimal car c'est une technologie embarquée hautement intégrée avec une consommation d'énergie ultra-basse.

3.11. CONCLUSION

Dans ce chapitre, nous avons présenté les circuits logiques programmables de type FPGA dont nous précisons ici les principaux avantages :

- Le premier argument est évidemment la nouvelle possibilité de reconfiguration dynamique partielle ou totale d'un circuit, ce qui permet d'une part, une meilleure exploitation du composant, une réduction de surface de silicium employé et donc du coût, et d'autre part, une évolutivité assurant la possibilité de couvrir à terme des besoins nouveaux sans nécessairement revoir l'architecture dans sa totalité. L'un des points forts de la reconfiguration dynamique est effectivement de permettre de reconfigurer en temps réel en quelques microsecondes l'ensemble ou une partie du circuit, c'est à dire de permettre de modifier la fonctionnalité d'un circuit en temps quasi réel. Ainsi le même CLB pourra à un instant donné être intégré dans un processus de filtrage numérique d'un signal et l'instant d'après être utilisé pour gérer une alarme. On dispose donc quasiment de la souplesse et la flexibilité d'un système informatique,

avec la différence fondamentale d'avoir une configuration matérielle, ce qui est infiniment plus puissante.

✦ Le second argument est que ces circuits n'ont pas la vocation à concurrencer les super calculateurs, mais plutôt à offrir une alternative en fonction de critères comme l'encombrement, les performances et le prix, et sont de ce fait bien adaptés à des applications de qualité dans le domaine des systèmes ambulatoires.

✦ Enfin, il semble que de plus en plus fréquent, les concepteurs de circuits ASIC préfèrent passer par l'étape intermédiaire d'un FPGA ce qui est moins risqué économiquement, puis une fois que le modèle FPGA est mis au point, il est alors aisé de le retranscrire dans une architecture de type prédiffusé ou précaractérisé. Ce que tous les fondeurs de silicium savent pour en faire un circuit réellement personnalisé et confidentiel. Le FPGA n'étant évidemment pas un circuit très sécurisé sur le plan de la confidentialité puisqu'il suffit d'analyser le contenu de la ROM associée pour remonter à la schématique imaginée.

Chapitre 4

LES LANGAGES DE DESCRIPTION MATERIELLE ET METHODOLOGIE D'IMPLEMENTATION

4.1. Introduction :

Les logiciels de CAO (Conception Assistée par Ordinateur), apparus à la fin des années 1960, permettent aux électroniciens de vérifier le fonctionnement d'un circuit sans l'avoir jamais réalisé matériellement. Dans un environnement CAO, on dispose de divers outils pour passer du concept à la description physique des circuits [70]. Ces outils rassemblent les interfaces de saisie de schéma, les simulateurs, les interfaces graphiques, les bibliothèques de modèles, les outils de layout et de vérification, etc.

La description matérielle a connu une progression au fur et à mesure que les moyens informatiques et les technologies d'intégrations des transistors (de plus en plus denses) évoluaient.

4.2. Les étapes d'évolution de La description matérielle

Les langages de description Matérielle ont connu une évolution en quatre étapes pour arriver à maturité :

Première étape.

Au début, la conception de circuits intégrés était manuelle. On dessinait les composants à la main, sur un papier spécial (Mylar) avec des crayons de couleur. C'est *le dessin au micron*. Une telle technique limitait naturellement la complexité des dispositifs conçus. La complexité était également limitée par les performances de la technologie disponible chez les "fondeurs" (ceux qui fabriquent véritablement le dispositif).

A cette époque le concepteur manipulait des objets élémentaires qui étaient des surfaces rectangulaires de couleur. Un transistor était constitué de l'assemblage d'une dizaine de ces rectangles (25 rectangles pour l'inverseur CMOS ci-dessous).

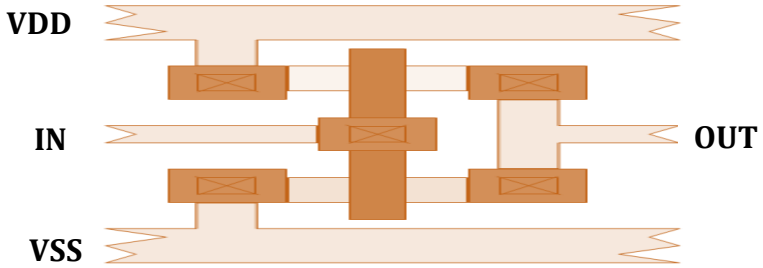


Figure 4.1 : Dessin au micron d'un inverseur CMOS

Deuxième étape.

Lorsque la technologie a évolué suffisamment pour offrir des ordinateurs puissants et des technologies permettant d'intégrer un grand nombre de transistors sur la même puce, il a fallu envisager de nouvelles méthodes de travail, comme les *langages de description de matériel*

(HDL). Ils avaient pour but de *modéliser*, de *simuler*, mais aussi de *concevoir*. Des outils informatiques permettant de traduire automatiquement une description textuelle en dessins de transistors (dessins au micron) ont fait leur apparition.

Les principes de cette évolution-révolution sont assez simples :

- Les langages sont *structurels*, c'est à dire qu'ils décrivent la structure de l'ensemble par assemblage de composants élémentaires.
- Il existe des bibliothèques de composants élémentaires. Ces bibliothèques contiennent, pour chaque composant élémentaire, un dessin au micron, destiné au placement-routage, et une représentation de la fonction réalisée, destinée au simulateur.
- Les outils informatiques (simulateurs, placeurs-routeurs) utilisent la description textuelle structurelle de l'ensemble et le contenu des bibliothèques.

Le concepteur est donc déchargé d'une partie importante du travail. Il aborde les problèmes à un niveau d'abstraction plus élevé. Il manipule des objets élémentaires de l'ordre de la dizaine de transistors : les portes logiques. Un exemple de description textuelle est présenté ci-dessous :

```
Bloc BK    (IO,  I1,   I2 :   in ;
           S0,   S1 :  out )
     Nœud      N;
     {
     inv (I0,      N);
     nand (I1,     I2,    S1);
     And (N,       S1,    S0)
     }
```

Figure 4.2 : Exemple de description textuelle

Troisième étape.

Cette étape a connu l'apparition des interfaces graphiques et donc des éditeurs de schémas simplifiant le travail de conception de circuits intégrés. En effet : il est en général beaucoup plus facile de lire et de comprendre un schéma qu'une description textuelle. La description textuelle est remplacée par une description schématique**.**

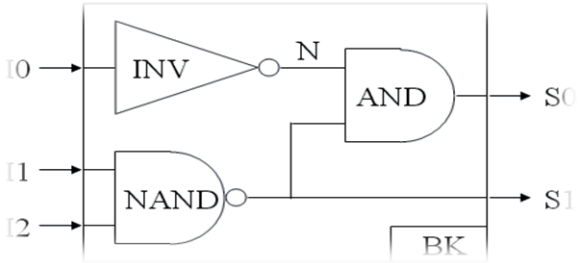


Figure 4.3 : Description schématique du même circuit

Il est important de comprendre en quoi cette modification, en apparence fondamentale, n'est en fait que superficielle. Les anciens outils informatiques restent inchangés, à ceci près qu'on leur adjoint un traducteur automatique de description schématique en description textuelle. La seule réelle modification est d'ordre ergonomique : il est en général beaucoup plus facile de lire et de comprendre un schéma qu'une description textuelle.

Les deux exemples ci-dessus montrent l'équivalence entre les deux types de représentation. On imagine aisément comment passer de la forme schématique à la forme textuelle. On comprend aussi pourquoi les informations supplémentaires contenues dans la première (symboles des portes, position relative et orientation des portes, taille et couleur des objets, polices de caractère utilisées, etc.) ne sont utiles qu'au seul concepteur humain. Les outils informatiques n'en ont pas l'usage.

Certaines propriétés "intéressantes" des langages textuels de description de matériel méritent d'être notées :

- La portabilité est accrue : Une description textuelle est plus facilement transportable d'un outil de CAO à l'autre, d'un concepteur à l'autre, d'une entreprise à l'autre.
- La maintenabilité est plus grande : Il est beaucoup plus facile de modifier une description textuelle qu'un schéma.
- On peut facilement archiver un module sous la forme de sa description textuelle afin de le réutiliser dans un autre projet (éventuellement après modification). C'est plus difficile avec d'autres représentations pour lesquelles le coût de mise à jour est plus élevé.

Quatrième étape.

Les langages *fonctionnels* (ou *comportementaux*) de description de matériel, grâce aux nouvelles possibilités de description à un niveau d'abstraction plus élevé, ont répondu à des besoins fondamentaux des concepteurs de circuits intégrés :

- La réduction des temps de conception.
- L'accélération des simulations qui devenaient prohibitives avec l'accroissement de la complexité des dispositifs.
- La normalisation des échanges : des langages normalisés et universellement reconnus servent aux échanges entre partenaires industriels, entre fournisseurs et clients.
- L'anticipation : grâce aux modèles HDL, il est possible de concevoir un système alors que ses composants ne sont pas encore disponibles.
- La fiabilité : les langages HDL sont conçus pour limiter en principe les risques d'erreur.
- La portabilité : les langages normalisés sont très largement portables.
- La maintenabilité et la réutilisabilité : les modifications et adaptations sont rendues plus simples, donc moins risquées et moins coûteuses.

La complexité des circuits conçus augmente toujours. On conçoit aujourd'hui des circuits composés de l'assemblage de *plusieurs millions de portes*. On a donc tiré encore une fois parti de l'augmentation de puissance des ordinateurs et des progrès réalisés en informatique pour imaginer des outils logiciels encore plus puissants : les *synthétiseurs logiques* [71].

Il s'agit de décharger les concepteurs d'une tâche de plus, afin de leur permettre de travailler à un niveau d'abstraction encore plus élevé, de manipuler des objets élémentaires encore plus grands. Pour ce faire, la description structurelle ne suffit plus. En effet, le nombre de primitives utilisables ne doit pas dépasser quelques centaines; au-delà, le concepteur ne peut plus mémoriser le contenu des bibliothèques et donc ne parvient pas à en tirer le meilleur profit.

Pour aller encore plus loin, il faut permettre au concepteur de décrire le "quoi" au lieu du "comment". Il doit pouvoir s'affranchir de la connaissance des primitives disponibles. Le dispositif à modéliser ou à concevoir sera désormais représenté par sa *fonction* et non plus par sa *structure*. C'est le synthétiseur logique qui déterminera la structure automatiquement à partir de la fonction. Bien sûr, les possibilités de description structurelle existent toujours mais on leur associe d'autres possibilités afin d'augmenter l'efficacité de l'ensemble.

```
Bloc  BK (IO,      I1,    I2:    in;
          S0,    S1:    out)
      {
      S0 = non (I0   ou I1etI2));
      S1 = non (I1) ou non   (I2);
      }
```

Figure 4.4 : Description fonctionnelle (comportementale) du même circuit

Les langages *fonctionnels* de description de matériel possèdent des avantages certains sur les langages structurels en terme de portabilité, maintenabilité et versatilité.

4.3. Utilité des HDL :

Il est important de comprendre la nature des différentes utilisations que l'on peut faire des langages fonctionnels de description de matériel.

4.3.1. *Simulation* : Le but de la modélisation, c'est la simulation. Il existe deux points importants pour la simulation :

❖ La fidélité : un modèle se doit d'être aussi précis que possible dans son champ d'application.

❖ L'efficacité : le modèle doit pouvoir être simulé le plus rapidement possible et doit être portable, réutilisable et facile à maintenir.

Pour simuler un modèle, il faut disposer du modèle, bien sûr, mais aussi de stimuli, c'est-à-dire de la description des signaux d'entrées du modèle au cours du temps. Ces stimuli sont appelés les TESTSBENCHES [72].

Les langages fonctionnels de description de matériel permettent de simplifier la conception en analysant automatiquement les résultats en cours de simulation [73].

Un environnement de simulation complet comprend : un générateur de stimuli, un modèle de l'objet à simuler et un vérificateur automatique des résultats. Ces trois composantes sont modélisées à l'aide du même langage.

4.3.2. Synthèse : Les langages fonctionnels de description matérielle servent aussi à concevoir. Il ne s'agit plus de modéliser en vue de la simulation, mais de décrire les objets qui seront véritablement fabriqués [74]. Si les considérations de vitesse d'exécution en

simulation existent toujours (la description sera simulée avant d'alimenter le synthétiseur, afin de vérifier que la fonction décrite est bien la fonction désirée) elles ne sont plus prioritaires. Ce qui compte le plus, c'est l'efficacité du code au sens du synthétiseur. En effet, pour que ce dernier produise la description structurelle la plus économique possible (et donc la surface de silicium la plus petite possible) [71].

4.4. Exemples de langages de description matérielle HDL

Il existe un grand nombre de HDL, les principaux étant :

- VHDL
- Verilog
- System C, sur-ensemble de C++ possédant des classes spéciales pour modéliser le matériel, et incluant un moteur de simulation.
- Confluence, langage déclaratif GPL pouvant générer du Verilog, du VHDL, des modèles exécutables en C et des modèles de vérification formelle.
- ABEL ("Advanced Boolean Expression Language", un langage propriétaire développé par Data I/O Corporation, maintenant possédé par Lattice).
- AHDL (Altera HDL, langage propriétaire d'Altera pour la programmation de leurs FPGA).
- CUPL (langage propriétaire de Logical Devices, Inc.).
- VHDL-AMS : extension de VHDL destinée aux circuits analogiques ou mixtes.

Les deux standards de langages de description de matériel les plus utilisé sont :

- VHDL
- Verilog

Leurs syntaxes sont assez différentes mais mettent en œuvre les mêmes concepts.

4.4.1 VHDL (Very High Speed Integrated Circuits Hardware Description Language) :

VHDL (VHSIC Hardware Description Langage) est un langage de description de matériel, c'est-à-dire un langage utilisé pour décrire un système numérique matériel, comme, par exemple, un flip-flop (bascule D) ou un microprocesseur. Il peut modéliser un système par n'importe quelle vue, structurelle ou comportementale, à tous les niveaux de description.

De plus il peut servir non seulement à simuler un système mais aussi à le synthétiser, c'est-à-dire être transformé par des logiciels adaptés (synthétiseurs) en une série de portes logiques prêtes à être gravées sur du silicium.

VHDL est l'un des trois grands langages de description de matériel utilisés majoritairement dans l'industrie.

À l'origine, avant même la mise en place du VHDL, le programme VHSIC (Very High Speed Integrated Circuits), impulsé par le département de la défense des états unis dans les années 1970-1980, a donné naissance au langage standard IEEE VHDL qui a été développé par le Groupe d'Analyse et de Standardisation VHDL (VASG, pour "VHDL Analysis and Standardization Group") [75]. La société CLSI (CAD Langage Systems Inc.), a préparé une série d'analyses et de recommandations dont a été tirée en février 1986 la version 7.2 de VHDL, point de départ du futur standard. La collaboration de CLSI au projet était financée par un contrat passé avec l'Air Force Wright Aeronautical Laboratories. Le standard définitif a été adopté vers le milieu de l'année 1987. La dernière version du standard (VHDL-2008) [76].

4.4.2. Verilog :

Verilog a été inventé par Gateway Design Automation Inc. aux alentours de 1984. C'était un langage propriétaire. Conçu à l'origine pour être utilisé dans leurs simulateurs logiques, mais le succès grandissant du VHDL a incité ses concepteurs de faire de Verilog un standard ouvert; c'est le standard IEEE 1364 dont il existe plusieurs versions, qui ont été enrichies pour offrir des fonctions équivalentes à celles de VHDL. La syntaxe de Verilog est réputée largement inspirée du langage de programmation C, bien que la ressemblance se limite aux expressions [77]. Ceci explique en partie son succès et sa diffusion rapide dans la communauté des ingénieurs qui ont déjà appris le langage C.

4.4.3. Comparaison entre VHDL et VERILOG et choix :

En comparaison entre VHDL et VERILOG, ils sont conçus pour répondre aux mêmes exigences des descriptions HDL, mais il se trouve que ces deux langages ont été conçus chacun en donnant plus d'importance à un aspect plutôt qu'un autre d'une description matériel. En effet, le langage VHDL se base plus sur le fait d'obtenir le circuit le plus optimisé que possible donc le moins encombrant dans le souci d'embarquer le plus de fonctionnalités dans un circuit logique programmable. Ce qui n'est pas sans conséquence car le code se voit plus long que celui rédigé en langage VERILOG pour une même fonction. Par contre, le langage VERILOG cherche à réduire le temps de mise en place d'un code opérationnel donc le plus court possible mais en réduisant les dimensions du code, en conséquence, le circuit synthétisé consomme plus de ressources. Ce qui peut apparaitre comme une contrainte, mais en réalité un circuit logique programmable rempli à 50% donne satisfaction pour les concepteurs et celui rempli à 80% est très optimisé [78].

C'est une des plus évidentes différences entre le Verilog et le VHDL.

- Le Verilog est beaucoup plus permissif que le VHDL... ceci comporte des bons et des mauvais côtés.

❖ **VHDL**

```
Signal  a       stdlogic (3 down to 0);
Signal  result  stdlogic (3 down to 0);
Result  <=      a + 1;    -- marche pas
Result  <=      a + '1'   -- marche pas
Result  <=      a + "0001"; -- marche
```

❖ **Verilog**

```
Reg [3:0] a;
Result  <=      a + 1;              // marche aussi
Result  <=      a + 1 'b1;          // marche aussi
Result  <=      a + 4'b0001;        // marche aussi
Result  <=      a + 25 ;            //marche aussi!
```

Figure 4.5 : Exemple VHDL/VERILOG

Dans la majorité des cas, le choix entre le VHDL et le VERILOG dépend de plusieurs facteurs

– La compagnie pour laquelle vous travaillez.
– Le client.
– La base de code existante.
– Les outils.
– Le prix des licences varie en fonction du langage.
– La région aussi, par exemple, le VHDL est plus populaire en Europe et en Afrique, moins aux USA et en Asie [79].
– Les écoles, universités.
– Maintenant ils existent des outils permettant de mélanger les deux.

4.5. Le langage de description matériel VHDL :

Le VHDL est un langage de description du matériel utilisé en électronique numérique. En tant que standard, il est indépendant du logiciel utilisé pour la compilation, la programmation des composants, la simulation. Il répond à tous les critères établis pour un langage HDL (voir paragraphe 4.2 Quatrième étape).

Les points forts de ce langage sont :

L'électronicien a toujours utilisé des outils de description pour représenter des structures logiques ou analogiques. Le schéma structurel que l'on utilise depuis si longtemps et si souvent n'est en fait qu'un outil de description graphique. Aujourd'hui, l'électronique

numérique est de plus en plus présente et tend bien souvent à remplacer les structures analogiques utilisées jusqu'à présent. Ainsi, l'ampleur des fonctions numériques à réaliser nous impose l'utilisation d'un autre outil de description [76].

Le deuxième point fort du VHDL est d'être un langage de description de haut niveau. D'autres types de langage de description, comme l'ABEL par exemple, ne possèdent pas cette appellation. En fait, un langage est dit de haut niveau lorsqu'il fait le plus possible abstraction de l'objet auquel ou pour lequel il est écrit. Dans le cas du langage VHDL, il n'a jamais fait référence au composant ou à la structure pour lesquels on l'utilise. Ainsi, il apparaît deux notions très importantes :

- ***Portabilité :*** avec deux objectifs, portable vis-à-vis du circuit logique programmable, c'est-à-dire peut-être implémenté sur n'importe quel circuit logique programmable à condition d'avoir la capacité logique requise. Portable vis-à-vis du compilateur, pouvoir passer d'un compilateur à un autre et obtenir un même circuit en fin de processus.
- ***Conception de haut niveau :*** c'est-à-dire qui ne suit plus la démarche descendante habituelle (du cahier des charges jusqu'à la réalisation et le calcul des structures finales) mais qui se "limite" à une description comportementale directement issue des spécifications techniques du produit que l'on souhaite obtenir.

Simulation, synthèse : le VHDL permet d'avoir des lignes de code pouvant être simulé (simulation fonctionnelle) dans le but de vérifier si le code obéit correctement à la fonction souhaitée mais pour parvenir à une maquette réalisable ce n'est pas suffisant d'où la nécessité de pouvoir synthétiser le même code. Lors de la synthèse, le compilateur traduit le premier code en un autre équivalent mais à un niveau d'abstraction plus bas. À ce niveau de synthèse, le compilateur traduit le code de haut niveau en portes logiques ceci en fonction du circuit logique programmable qui est ciblé. Le fichier synthétisé lui aussi, se doit d'être simulé (simulation événementielle) et vérifier si le compilateur (synthétiseur) est resté fidèle aux exigences de départ. Puis le fichier est compilé une deuxième fois pour aboutir au circuit logique qui sera implémenté. Cette fois encore, une troisième simulation (facultative) (simulation temporelle) pour être certain que le circuit est resté inchangé.

Une construction hiérarchique : cette approche permet de simplifier la conception puis ce que les taches peuvent être divisées pour aboutir au point le plus simple que possible puis faire un assemblage des blocs.

Une description fonctionnelle : complémentaire de la précédente, la vision fonctionnelle apporte la puissance des langages de programmation. Tout algorithme est la description interne d'un bloc situé quelque part dans la hiérarchie du schéma complet. La vision structurelle est concurrente, la vision algorithmique est séquentielle au sens informatique du terme. Un programme VHDL doit être compris comme l'assemblage en parallèle des taches indépendantes qui s'exécutent concurremment.

4.5.1. Structure d'un programme VHDL le couple ENTITY, ARCHITECTURE :

Un opérateur élémentaire, un circuit intégré, une carte électronique ou un système complet, est complètement défini par des signaux d'entrées et de sorties et par la fonction réalisée de façon interne. Les concepteurs du VHDL ont adopté l'approche suivante : n'importe quel système est considéré comme une boite noire. Cette boite noire a des entrées et des sorties. Ils ont appelé la boîte noire « ENTITY ». L'importe quel système électronique effectue des opérations sur le signal d'entré pour donner le résultat du traitement en sortie. Ces opérations sont le contenu de la boîte noire, ce contenu est appelé « ARCHITECTURE » [80].

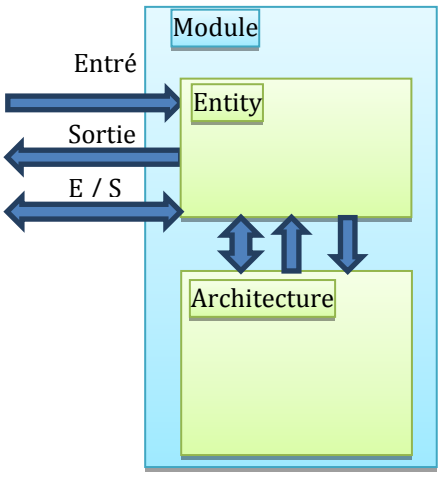


Figure 4.6 : structure de base d'un module sous VHDL

Entity ou Entité : Dans la partie déclarative de l'entité, le circuit est décrit comment il est vu par l'extérieur, ceci à travers les entrées, sorties et entrées/sorties.

```
Entity<entity name>is port (
        <signal name >     : <signal direction><data type>);
End<entity name>;
```

Figure 4.7 : Syntaxe déclarative de l'entité.

Architecture : l'architecture décrit le comportement que doit avoir le circuit ou les opérations qu'ils doivent effectuer. Une architecture se doit toujours être attachée à une

entité (ils vont de paire) [81].

```
Architecture< architecture name>of< entity name>is
<Define signals and constants>
Begin
End< architecture name>;
```

Figure 4.8 : Syntaxe déclarative de l'architecture.

C'est dans cette section que le programme est rédigé. Un programme comporte essentiellement les éléments suivants : les signaux internes, opérateurs logiques (synchrone ou concurrent), les process [81]. La description d'une architecture peut prendre trois formes :

Description comportementale : Ce type de description spécifie le comportement du composant ou du circuit à réaliser au moyen d'instructions séquentielles ou sous forme de flot de données (constituant un process).

Description structurelle : Dans ce type de description, les interconnexions des composants préalablement décrits sont énoncées. Cette description est la transcription directe d'un schéma.

Description mixte : Elle regroupe les deux descriptions décrites précédemment. À chaque entité peut être associée une ou plusieurs architectures mais au moment de l'exécution (simulation, synthèse…) seulement une architecture et une seule est utilisée.

4.5.2. Éléments du langage VHDL :

Le signal **:** le signal est représenté par certains types de données. Il est assigné par un nom et un type de donnée comme suit :

Signal « nom_signal » : « type_donné » :

Le terme signal indique implicitement au compilateur de convertir le signal en un canal d'interconnexion.

Les constantes **:** la constante est un objet qui porte une valeur à l'initialisation, l'enregistre tout au long du déroulement du programme. Elle est déclarée par le terme **Constant**.

Les variables **:** la valeur d'une variable est immédiatement mise à jour lorsqu'elle est assignée. Une variable peut être déclarée soit à l'intérieur d'un processus ou d'un sous-

programme et elle est locale à son bloc de déclaration. Une variable n'est utilisée que dans le domaine de la programmation séquentielle. Une variable est déclarée comme :

Variable « nom_variable » : type « expression initiale » ;

***Les types de données* :** le VHDL est un langage très typé. Chaque élément utilisé (signal, variable) doit avoir un type bien défini. Toute opération doit se faire avec des objets d'un même type. Sous VHDL les types de données sont standardisés et on y réfère par std_logic, std_logic_vector, bit, bit_vector… Ils sont regroupés dans les bibliothèques par défaut. L'utilisateur peut créer lui-même un type de donnée et l'intégrer dans une bibliothèque. Le tableau suivant montre les valeurs admises par le type standard logic_vector.

Value	Description
0	Low or logic zero
W	Weak unknown signal
L	Weak low
H	Weak high
U	Unknown or uninitialized
Z	High impedance
X	unknown
-	Don't care

Tableau 4.1 : les valeurs admises par le type standard logic_vector

***Le Process* :** il permet de décrire les instructions parallèles qui utilisent les mêmes signaux. Un process peut être vu comme un sous-programme en boucle infinie qui ne s'arrête qu'à l'instruction de synchronisation **wait** (attendre). L'énoncé process est habituellement accompagné d'une liste de sensibilité (horloge, bit d'activation ou de sélection…) et il est exécuté lorsqu'un des signaux de la liste de sensibilité change d'état.

***Les instructions concurrentes* :** les instructions concurrentes sont décrites directement dans le corps de l'architecture. L'objectif de ces relations étant d'affecter des valeurs à des composants ou de réaliser des connexions, alors elles n'ont pas l'ordre d'exécution. Elles sont exécutées en parallèle.

***Les instructions séquentielles* :** les instructions séquentielles sont internes aux processus, aux procédures et aux fonctions (sous-programmes). Elles constituent les outils de base des descriptions comportementales. Ces instructions, pour la plupart, sont inspirées des langages

classiques de programmation. Parmi elles, on retrouve **if-then-else** qui permet de réaliser les boucles conditionnelles. **Case-then** pour tester des valeurs et choisir l'opération à effectuer.

L'instruction **wait**, qui permet de suspendre l'exécution d'un processus jusqu'à ce que la durée spécifiée soit évaluée. Les boucles telque : **for-loop, while-loop**…

Les attributs **:** ce sont des propriétés ou des caractéristiques associées à un objet ou un type. Ils sont soit prédéfinis ou définis par l'utilisateur. Ils sont déclarés comme suit :

Nom_var' nom_attribut;

Les Librairies et les Paquetages **:** ce sont les librairies qui font qu'un programme est transportable d'un programme à un autre et permettent aussi leurs réutilisations. Une librairie est constituée d'un ou plusieurs paquetages et ce sont ces derniers qui renferment les fonctions, les procédures, les constantes, les types… elles sont fournies soit par l'IEEE en tant que partisan du standard VHDL ou bien créé par le concepteur lui-même [83]. La syntaxe déclarative se fait comme suit :

Library IEEE ; pour déclarer la librairie

Use IEEE.STD_LOGIC_1164.ALL ; pour choisir le paquetage Standard Logic1164

4.5.3. Structure d'un programme sous VHDL :

Un programme écrit sous VHDL obéit à la structure suivante :

Entête **:** c'est une partie facultative, elle referme des informations concernant le programmeur, la description du programme en général, la date de rédaction et toute information qui semblera importante pour celui qui rédige le programme.

Déclaration des librairies **:** en deuxième lieu vient la déclaration des librairies et des paquetages que le programmeur juge nécessaire pour son programme.

Déclaration d'entité **:** en troisième lieu vient la déclaration d'entité avec les signaux à utiliser dans tout le programme avec leur direction.

Déclaration d'architecture **:** juste après l'entité vient la déclaration de l'architecture qui décrit l'entité. Dans cette partie les signaux internes sont déclarés et les variables propres à cette même architecture, puis le programme est rédigé. Voir la Figure 4.9.

```
Library IEEE;
Use IEEE.std_logique_1164.all;
Entity<entity name>is port
        (<list of ports or design inputs and outputs>);
End<entity name>;
Architecture< architecture name>of< entity name>is
        < In this section define signals and constants>
Signal <signal name>      : Data type;
Begin
< Concurrent statements>
<Process name>  : process ( sensitivity list)
Begin
<sequential statements>
End ;
End< architecture name>;
```

Figure 4.9 : Structure d'un programme sous VHDL [84].

4.6. Les TESTBENCHS :

La plupart des simulateurs logiques permettent de créer des stimuli pour vérifier le comportement d'un modèle. Souvent un langage de commande donne à l'utilisateur la possibilité d'automatiser, par la création de fichiers contenant des instructions de ce langage, l'enchaînement des opérations nécessaires aux tests. Le langage VHDL est un langage de modélisation et de simulation. Il contient tous les éléments nécessaires à la création de stimuli et surtout à l'exploitation des résultats.

Pour élaborer un TESTBENCH, il faut établir la liste des cas la plus complète que possible. Les stimuli sont les entrées appliquées au programme via le simulateur pour limiter le comportement des vraies entrées. Après exécution du simulateur, les sorties sont observées si elles sont comme prévu par le programme et sans aucun risque sur le matériel (FPGA…) [85].

4.7. Méthodologie de conception :

4.7.1. Les outils de CAO pour la configuration d'un FPGA :

Le rôle principal confié aux outils de CAO se résume en 4 étapes qui sont : la description, la simulation, la synthèse, le placement et le routage et en dernier la configuration du FPGA. Un design peut être conçu à l'aide d'un éditeur schématique lors de la conception des circuits simples ou d'un outil de programmation utilisé pour les circuits complexes [86].

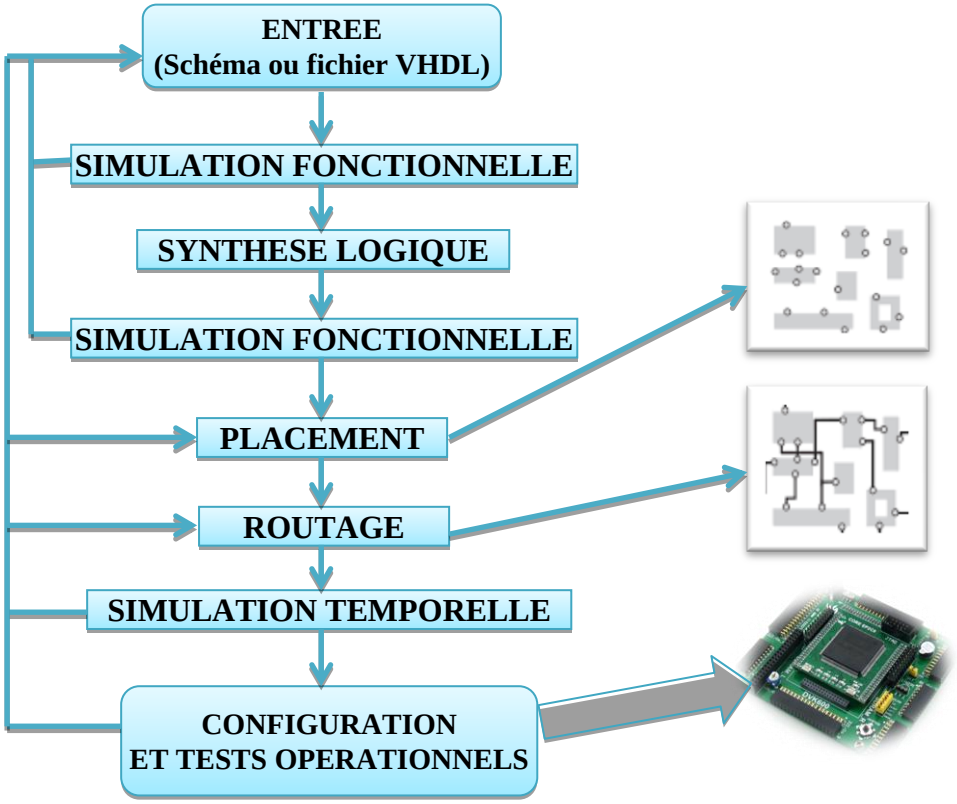


Figure 4.10: Mode d'exécution matériel des outils de CAO.

4.8. Etapes de conception sur FPGA.

Le flot de conception d'un système sur puce regroupe plusieurs niveaux d'abstraction.
Dans chaque niveau, le concepteur s'intéresse à la résolution d'un problème. Les outils de *CAO* sont utilisés intensivement et assurent la transition entre les différents niveaux d'abstraction. Nous pouvons traiter un système complexe de deux manières qui sont :

L'approche dite *« descendante » (ou « **top-down** » en anglais).*

L'approche dite *« ascendante » (ou « **bottom-up** » en anglais).*

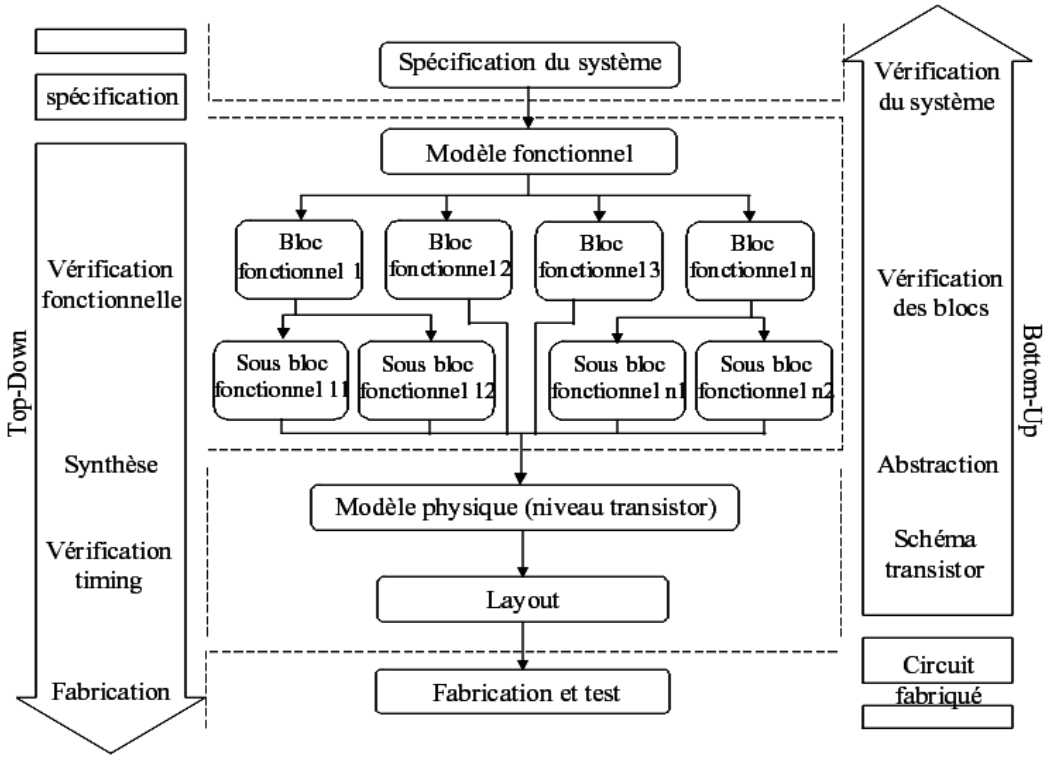


Figure 1.11: Méthodologie de conception hiérarchique Top-Down et Bottom-Up [87].

Le développement d'une application sur *FPGA* par des outils *CAO* suit l'enchainement des étapes suivantes :

4.8.1. Spécification du design

Le nombre de broches d'entrée-sortie et leur localisation dans la puce *FPGA*.

La spécification de la fréquence d'horloge du système.

La spécification de la mémoire requise pour l'application.

4.8.2. Développement du design

Spécification de la méthodologie de design (Outil de développement utilisé).

La saisie du circuit Codage *RTL* (*VHDL, Verilog*...)

Graphique *(Machine à états).*

Saisie HDL *(**H**ardware **D**escription **L**anguage).*

La simulation (Pré et Post synthèse).

4.8.3. Synthèse

La synthèse est le processus qui convertit la représentation du design à partir du code *HDL* fourni pour produire une représentation au niveau porte logique [88]. Elle cherche à déterminer quelles sont les structures susceptibles de répondre au cahier des charges étudié et

de produire un code booléen unique sous forme d'un fichier.

4.8.4. Placement et routage

A partir des fichiers de synthèse, l'outil de conception procède au placement et routage. Un algorithme de routage est sensé faire l'aiguillage des données qu'il reçoit vers leurs destinations par action sur les nœuds de routage ce qui est équivalent à définir les chemins qui relient l'ensemble des ***CLB*** contenus dans la fonction désirée. Ces algorithmes de routage sont différents d'un concepteur à un autre. Plusieurs traitements sont nécessaires pour obtenir un fichier de configuration :

Partitionnement* :** Les équations logiques sont partitionnées en un autre ensemble équivalent d'équations. Chaque équation de ce nouvel ensemble peut être implantée dans un seul bloc logique du composant cible ***FPGA.

***Placement* :** Des blocs logiques sont sélectionnés dans la matrice et affectés au calcul des nœuds du réseau booléen.

Routage : Les ressources d'interconnexion sont affectées à la communication de l'état des nœuds du réseau vers les différents blocs logiques qui en ont besoin.

Génération des données numériques de configuration : Les informations abstraites de routage, de placement et les équations implantées dans les blocs sont transformées en un ensemble de valeurs numériques, qui seront chargées sur le composant ***FPGA***.

4.8.5. Intégration et implémentation

L'implémentation est la réalisation proprement dite qui consiste à mettre en œuvre l'algorithme sur l'architecture du circuit configurable cible, c'est-à-dire à compiler, charger, puis lancer l'exécution sur un ordinateur ou calculateur. C'est une étape de programmation physique et de tests électriques qui clôture la réalisation du circuit. La figure suivante résume un l'ensemble de ces étapes.

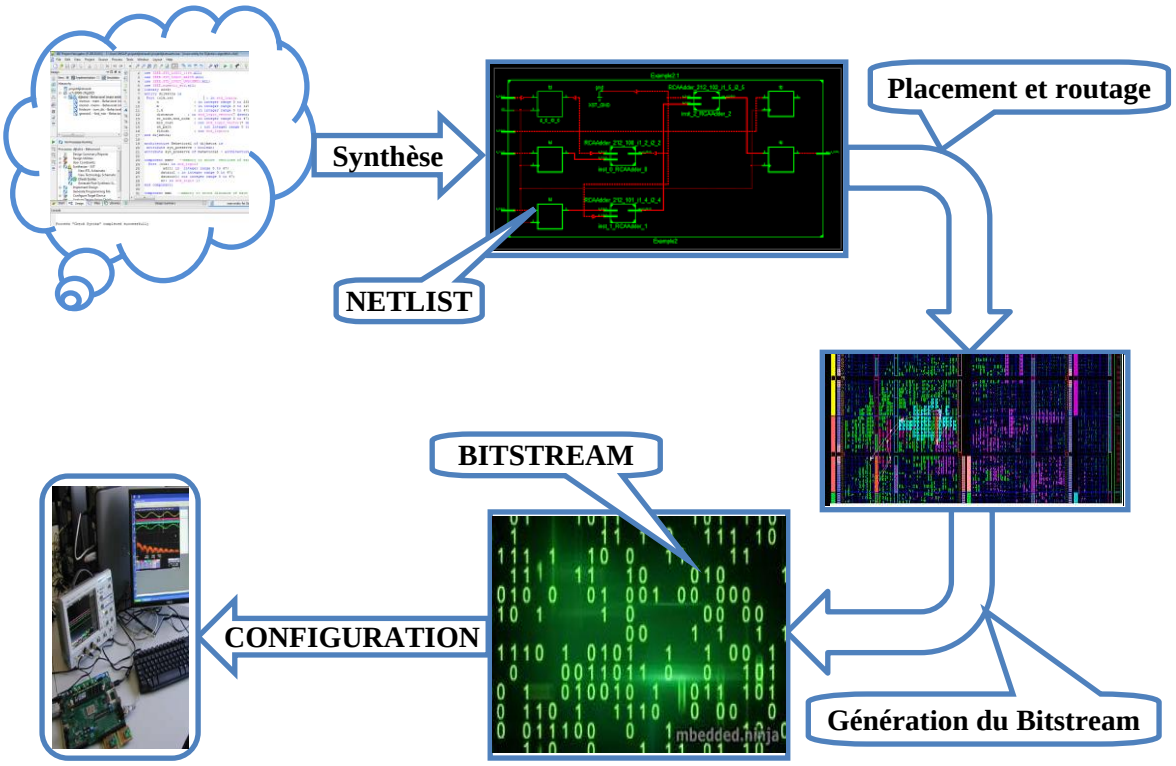


Figure4.12 : Les étapes de conception sur FPGA [86].

4.9. Conclusion :

La conception assistée par ordinateur est maintenant reconnue dans tous les domaines. Toutefois, il est nécessaire de continuer à la promouvoir, pour que la CAO ne soit plus uniquement utilisée comme un simple outil de dessin. Les systèmes actuels permettent la gestion des connaissances et des maquettes numériques qui doivent être utilisées afin de valoriser le système.

Le VHDL est un langage qui peut déconcerter, au premier abord, le concepteur de systèmes numériques, plus habitué aux raisonnements traditionnels sur des schémas peu familiers des langages de description abstraite. Il est vrai que le langage est complexe. Mais ce type d'approche est d'une très grande souplesse, et d'une efficacité redoutable. Des problèmes de synthèse qui pouvaient prendre des heures de calcul, dans une démarche traditionnelle, sont traités en quelques lignes de programme.

Chapitre 5

RESULTATS ET DISCUSSION

5.1. Introduction

Comme cité dans le chapitre 1, le routage classique dans les réseaux de télécommunication ne répond pas aux exigences des futurs systèmes qui intègrent un très grand nombre de routeurs avec la QoS. Le processeur FPGA est un nouveau paradigme de routage pour les réseaux de télécommunication. Il a été proposé comme solution prometteuse pour résoudre les problèmes rencontrés au niveau des processeurs classiques et actuels [89].

Ce travail permet de concevoir à partir du langage VHDL, un modèle équivalent à l'algorithme de Dijkstra, sachant qu'un modèle d'un circuit n'est qu'une abstraction de son comportement. Ce modèle va nous permettre de tester le routage de l'information dans les réseaux de télécommunication et de nous assurer de la validité de chaque partie grâce à l'utilisation d'un logiciel de simulation intégré dans l'environnement CAO. Il est fondamental d'apporter la preuve mathématique du bon fonctionnement du circuit ou bien d'émuler son fonctionnement. La simulation du système est faite pour différentes raisons non seulement pour vérifier la validation du code mais aussi un outil d'analyse permettant de prévoir le comportement du système sous l'action d'un événement particulier et la visualisation de l'évolution temporelle [90]. La dernière phase de cette conception consiste à faire migrer cet algorithme sur un démonstrateur matériel FPGA afin de vérifier sa faisabilité technique et d'évaluer ses performances. Donc, ce chapitre s'occupe de la conversion vers un format exécutable (langage de description matériel) de l'algorithme de Dijkstra puis la vérification du circuit intégré correspondant à cet algorithme en utilisant l'outil ISE 14.2 de Xilinx.

5.2. Description de l'architecture en VHDL

On est sensé élaborer une plate-forme matérielle dédiée au routage de l'information dans les réseaux de télécommunication basée sur des composants du type « ***S**ystem **O**n **C**hip* » ou « ***S**ystem **O**n **P**rogrammable **C**hip »*, entièrement portable puisqu'elle sera décrite en langage VHDL qui est indépendant vis-à-vis de la technologie matériel cible. Comme vu dans le chapitre précédent, il existe différentes manières de faire décrire un circuit numérique sur différents niveaux d'abstraction (la description comportementale, la description structurelle, la description bas-niveau). On peut mélanger des blocs de niveaux différents pour la simulation dans certains outils les plus élaborés.

Pour implémenter une spécification HDL sur un FPGA, avec la platef orme ISE14.2, quatre étapes importantes (expliquées dans le chapitre précédant), sont illustrées par la Figure 5.1.

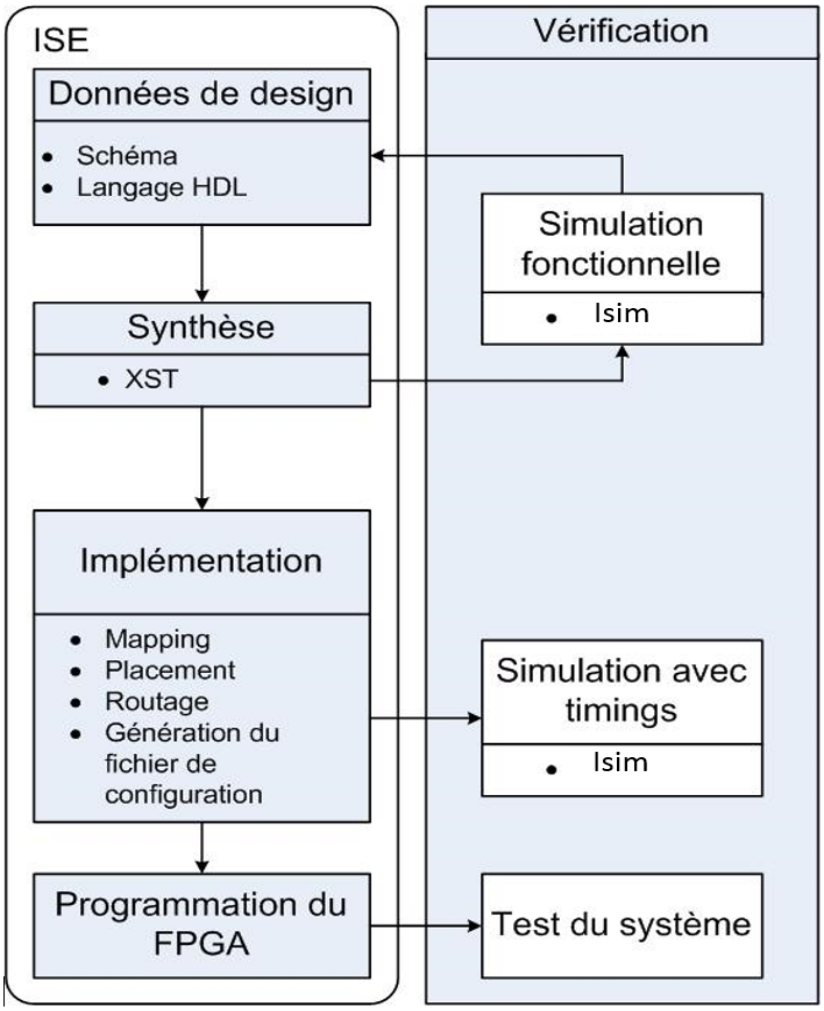


Figure 5.1 : Les étapes d'implémentation d'un circuit sur FPGA sous l'outil ISE

Ces étapes sont communes à tous les fabricants de circuits logiques programmables. Chaque fabricant établit un schéma en fonction de ses produits proposés et des logiciels offerts pour pouvoir implémenter une spécification HDL.

La démarche que nous avons suivie pour décrire et vérifier le circuit est résumée comme suit :

- Analyse et décomposition de l'architecture proposée.
- Description en VHDL et validation de chaque bloc.
- Rassembler les blocs élémentaires du système.

Nous avons modélisé chaque composant séparément et nous avons stocké les modèles dans des bibliothèques de composants réutilisables afin d'économiser le temps de développement. Des modules supplémentaires peuvent être ajoutés si nécessaire car le langage de description matérielle VHDL facilite la modification et la réutilisation d'un design.

5.3. Présentation de l'environnement de développement « Xilinx Integrated Software Environnement (ISE) »

Le logiciel ISE de XILINX (Integreted Software Environment) est un environnement de développement qui possède différents outils de CAO. Les sociétés spécialisées en CAO microélectronique fournissent des environnements logiciels spécialisés. Tous les fabricants de FPGA proposent des outils de CAO pour configurer leurs circuits (on a *ISE pour Xilinx et QUARTUS ou MAX + II pour ALTERA*) [91]. L'offre logicielle dans le domaine de conception des circuits numériques est très variée et l'une parmi ces environnements que nous allons exploiter au cours de ce travail à savoir XILINX ISE qui est un Logiciel de création et de gestion de projets CAO qui est un environnement de conception. C'est un logiciel multitâche qui possède dans son soft différents outils permettant la création de système sous circuits numériques. L'introduction de projets se fait de deux manières qui sont textuelle ou graphique en vue d'une intégration dans un circuit logique programmable (CPLD ou FPGA), sachant que la saisie graphique est une alternative à la saisie textuelle mais limitée. Ce logiciel Xilinx ISE permet la simulation de la description et la synthèse du circuit logique équivalent puis placer et router ce circuit sur un prototype correspondant à une technologie FPGA bien précise et enfin, lorsque toute les vérification sont faites, vient l'implantation sur un FPGA réel ce qui correspond à générer le fichier de configuration du circuit cible choisi afin d'établir les interconnexions des cellules logiques correspondantes au circuit logique conçu avec optimisation de ressources disponibles au niveau circuit programmable FPGA. D'une manière générale, le XILINX ISE permet de réaliser toutes les étapes de conception et de programmation des FPGA de XILINX et même pour d'autres circuits programmables tels que les CPLD.

La conception de circuits sur XILINX ISE met en œuvre quatre outils : un éditeur de texte ou entrée graphique, un simulateur, un synthétiseur et un placeur-routeur. L'éditeur de texte ou entrée graphique est pour faire introduire la description dans les logiciels CAO c'est-à-dire de dessiner ou décrire le circuit avec une interface graphique ou textuelle. La simulation du système est faite pour vérifier la validité du code avant-synthèse, après-synthèse et même après le placement et routage. Les deux étapes synthèse et routage succéderont par la suite où la synthèse consiste à faire la transcription de description d'une forme texte vers une autre forme graphique (RTL) à base de portes logiques et pour la deuxième étape nommée routage n'est qu'une adaptation du circuit logique synthétisé sur les ressources disponibles dans le circuit FPGA ciblé.

5.4. Présentation de la démarche suivie :

Dans ce travail de thèse nous avons proposé une nouvelle technique pour accélérer le processus de routage dans les réseaux de télécommunications et en gardant une meilleure prise en compte de la QoS dans les décisions de routage. Pour appuyer notre démarche, nous avons effectué une analyse critique sur les différents travaux de recherche qui avaient pour objectifs de développer et d'améliorer les performances des méthodes avec QoS. Cette recherche a été orientée vers des solutions qui se basent sur la conception de nouveaux protocoles et algorithmes de routage [92] [93] c'est-à-dire des solutions softs qui sont encore plus complexe, au moment où les chercheurs indiquent que le problème est au niveau du processeur qui effectue les calculs, d'où la nécessité de se tourner vers des nouvelles technologies plus performantes que celles rencontrées sur les processeurs actuels. Cette analyse nous a conduits à retenir comme base de travail, d'utiliser une technologie qui allie la flexibilité du software et la vitesse du hardware, c'est pour cela que nous avons choisi la technologie FPGA pour l'ajouter au routeur actuel et l'algorithme de DIJKSTRA pour la recherche du plus court chemin.

Nous avons remarqué que l'algorithme de Dijkstra (étudié dans le chapitre 2) mène à une solution pouvant comporter une ou plusieurs boucles. Pour pouvoir l'améliorer, on fait un mécanisme de test lors de l'ajout d'un nouveau nœud [94]. Pour chaque nœud i, on teste les nœuds du chemin parcouru en partant de i jusqu'à atteindre le nœud d'origine. Si le nœud existe déjà dans le chemin calculé, il sera éliminé de l'arbre d'exploration. La Figure 5.2 illustre la nouvelle formulation de la deuxième étape de l'algorithme de Dijkstra classique.

```
/* ETAPE 2 modifiée */
Si (counti<= K) alors                    (S'il n'y a pas trop de chemins passant par ce nœud)
    Début
        Pour eachar c(i,j) Є i           (Pour chaque arc (c'est à dire chaque lien)
                                          appartenant à ce nœud )
            Début
                                         (On vérifie que j n'est pas déjà dans le chemin calculé)
                v=k                      (Sauvegarde de l'étiquette du nœud traité)
                Tant que                 (h(v) != s) (Remonter le chemin jusqu'à la source)
                    Début
                        if (h(v) == j)   (Si le nœud correspond au nœud que
                                          l'on souhaite ajouter)
                        Début
                            goto ne_pas_ajouter
                        Fin
                        v =precedent_v(Remonter au nœud précédent)
                    Fin
                elem = elem + 1          (Ajouter le nœud situé à l'autre extrémité de
                                          l'arc, dans la liste des candidats)
                distance_elem = distance_k + c_ij   (Sauvegarde des informations
                                                     de la nouvelle étiquette)
                precedent_elem = k
                h(elem) = j              (Pour retrouver le nœud auquel appartient
                                          l'étiquette)
                h_-1(j) = h_-1(j) U {elem}
                X = X U {elem}           (Ajouter l'étiquette à la liste des candidats)
                Ne_pas_ajouter:
            Fin
    Fin
```

Figure 5.2 : la nouvelle formulation de la deuxième étape de l'algorithme de Dijkstra. [32]

5.4.1. Modélisation de la topologie réseau.

Les routeurs d'aujourd'hui qui exécutent le processus de routage éprouvent de plus en plus des difficultés croissantes avec l'augmentation de la taille des réseaux et les exigences de la qualité de service. En effet, l'amélioration en QoS aboutit à la complexité du calcul d'une manière exponentielle, donc il y a une exigence d'une augmentation en ressources opératoires surtout dans le cas des protocoles à état de lien pour que le réseau atteigne des performances acceptables. Afin d'améliorer la performance du réseau en termes de paramètres mentionnés précédemment, nous proposons une architecture de routage qui se base sur l'algorithme de Dijkstra modifié pour l'implémenter sur un processeur FPGA.

En fait, nous avons adapté le protocole OSPF qui utilise l'algorithme de Dijkstra pour trouver le chemin le plus court, ce protocole de routage permet à tous les routeurs de connaitre la capacité des liens à supporter. Pour la faisabilité de notre application, et comme nous montre la figure 5.3, chaque routeur établit des relations d'adjacence avec ses voisins immédiats en envoyant des messages *hello*. Chaque routeur communique ensuite la liste des réseaux auxquels il est connecté par des messages LSA (*Link-state advertisements)* propagés de

proche en proche à tous les routeurs du réseau. L'ensemble des LSA forme une Base de données topologiques de l'état des liens LSDB (*Link-State Database)* pour tout le réseau qui est identique à tous les routeurs participant dans ce réseau. Chaque routeur utilise ensuite l'algorithme de Dijkstra, *Shortest Path First* (SPF) pour déterminer la route la plus courte vers chacun des réseaux connus dans la LSDB.

Les informations qui sont échangées dans les différents paquets OSPF sont présentées dans le premier chapitre.

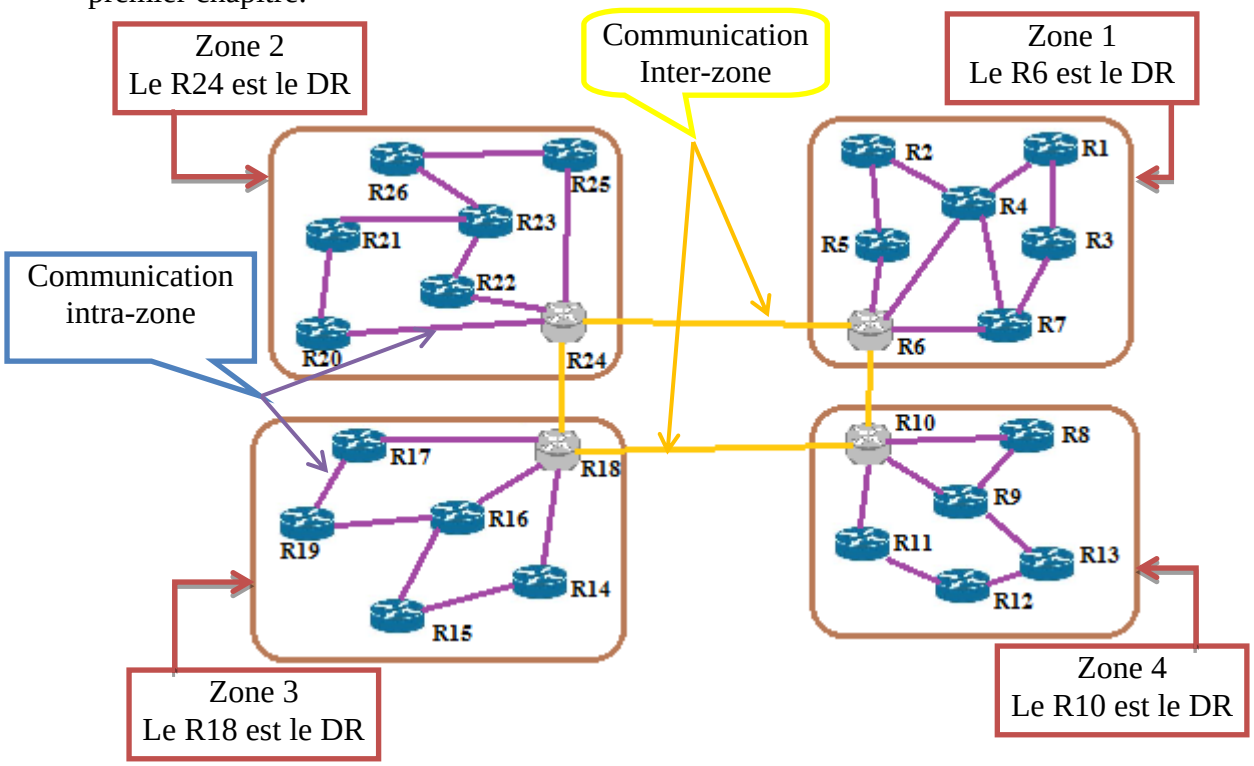


Figure 5. 3 : Modélisation du problème

Pour les réseaux importants qui intègrent un très grand nombre de routeur, on place l'ensemble des routeurs en groupes connexes appelés zones où les routeurs les plus communicants sont regroupés dans la même zone. Dans chaque zone on désigne deux routeurs, un primaire qu'on appelle DR (routeur désigné), l'autre secondaire BDR (Backup Designated Router). Il y a plusieurs avantages à cette architecture. En effet,

- ✓ Réduire le trafic lié à l'échange d'informations résultats des liens (car il n'y a pas d'échange entre tous les routeurs mais entre chaque routeur et le DR),
- ✓ Améliorer l'intégrité de la base de données topologiques (car cette base de données doit être unique),
- ✓ Accélérer la convergence et la vitesse de calcul des routes.

Dans cette démarche, nous proposons de remplacer le routeur désigné qui a été un processeur ordinaire par un routeur à base d'un processeur FPGA en utilisant l'algorithme de Dijkstra modifié pour calculer les chemins.

En effet, pour un paquet de données venant d'un poste de réseau et arrivant à l'entrée d'un réseau selon la métrique spécifiée, le routeur DR doit lui assigner le chemin optimal.

La topologie réseau choisie est constituée de deux types de routeurs dans chaque zone (figure 5.3) : un routeur FPGA pour chaque zone et les autres sont des routeurs ordinaires. Chaque routeur, que ce soit ordinaire ou FPGA a son identifiant appelé RID. Les routeurs DR utilisant une adresse multicast, cette adresse désigne tous les routeurs DR (par exemple dans le réseau Internet est 224.0.0.6), ce qui signifie que le DR et le BDR doivent être en écoute à cette adresse puis le DR relaie les mises à jour à tous les autres routeurs ordinaires en utilisant une adresse prédéfinie (224.0.0.5 pour le réseau Internet). Le BDR reçoit les mises à jour mais ne les partage pas, il se place juste près au cas où le DR tombe en panne.

5.4.2. Les étapes de notre démarche sont les suivantes :

En figure 5.4, on va illustre les étapes.

- **La première étape** : Dans un premier temps, lorsque le routage OSPF est lancé sur le routeur de départ, des paquets de données HELLO sont envoyés sur chaque interface où le routage dynamique a été activé. Ces messages HELLO permettent de calculer la densité de charge de communication entre chaque paire de nœuds pour une application choisie pour sélectionner les nœuds les plus communicants dans le réseau et construire la base d'adjacences.
- **La deuxième étape** : Après avoir obtenu la base d'adjacences, on peut mettre les nœuds les plus communicants dans la même zone; le nombre de nœuds dans chaque zone est limité. Dans notre cas, chaque zone est constituée de deux types de routeurs : un routeur ordinaire et un routeur FPGA (DR). La gestion des zones est effectuée de manière à obtenir un minimum de communication inter-zones (entre les différentes zones), où la zone « 0 » communique plus avec les zones « 1 » et « 2 » qu'avec la zone « 3 », la raison pour laquelle nous mettons la zone « 0 » au voisinage des zones « 1 » et « 2 » et loin de la zone « 3 ». La même opération est effectuée pour toutes les autres zones.
- **La troisième étape** : Dans cette étape, l'un des routeurs doit être élu « routeur désigné » (DR Designated Router) et un autre « routeur désigné de secours » (BDR Backup Designated Router). Les « routeurs désignés » (DR) contiennent des informations supplémentaires sur le routage dans tout le réseau. Le routeur désigné sert de référence

pour la base des données topologiques représentant le réseau. Le rôle du DR est de gérer les communications entre les membres de différentes zones. Comme vu plus haut, la sélection du DR se fait selon certains critères. Cependant, dans notre cas, Le routeur qui envoie un message Hello avec la plus grande priorité OSPF est élu DR.

- **La quatrième étape** : La dernière étape est la sélection de l'algorithme de routage. Notre algorithme de routage est constitué de deux niveaux : un niveau où les communications se font au sein de la même zone (intra-zone) et un autre où les communications se font entre les différentes zones (interzone). La figure suivante résume les étapes de notre démarche.

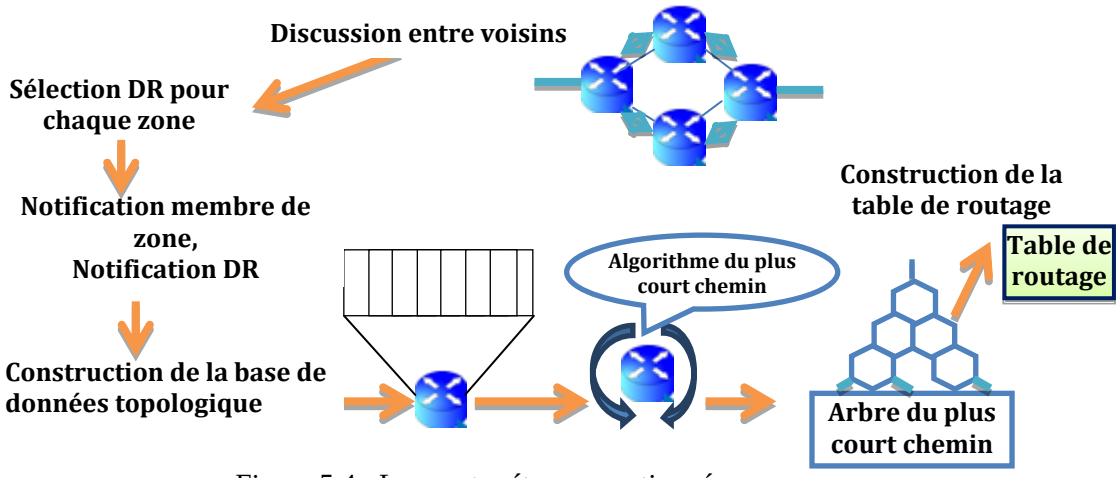


Figure 5.4 : Les quatre étapes mentionnées.

5.4.3. L'architecture de l'algorithme de Dijkstra proposée

L'architecture proposée dans ce travail consiste à calculer le plus court chemin en se basant sur l'algorithme de Dijkstra amélioré pour l'implémenter sur carte FPGA est représenté sur la Figure 5.5.

L'implantation de notre architecture sur FPGA consiste en la programmation du circuit à l'aide du langage de description de haut niveau VHDL. Nous allons donc décrire le circuit sous forme d'algorithme VHDL. La seconde étape consiste à effectuer la simulation. La compilation a été validée à l'aide de l'outil ISE 14.2 de la société Xilinx [95]. Ce logiciel fournit à la fin de ces phases, divers résultats sous forme de rapports et schémas. Parmi ces documents, le rapport de compilation (tableau5-2) nous donne les quantités de ressources logiques utilisées pour l'implémentation physique du système, tandis que la fonction RTL viewer (Register Transfert Level) présente une description hiérarchique par niveaux d'abstraction du système sous forme de schémas.

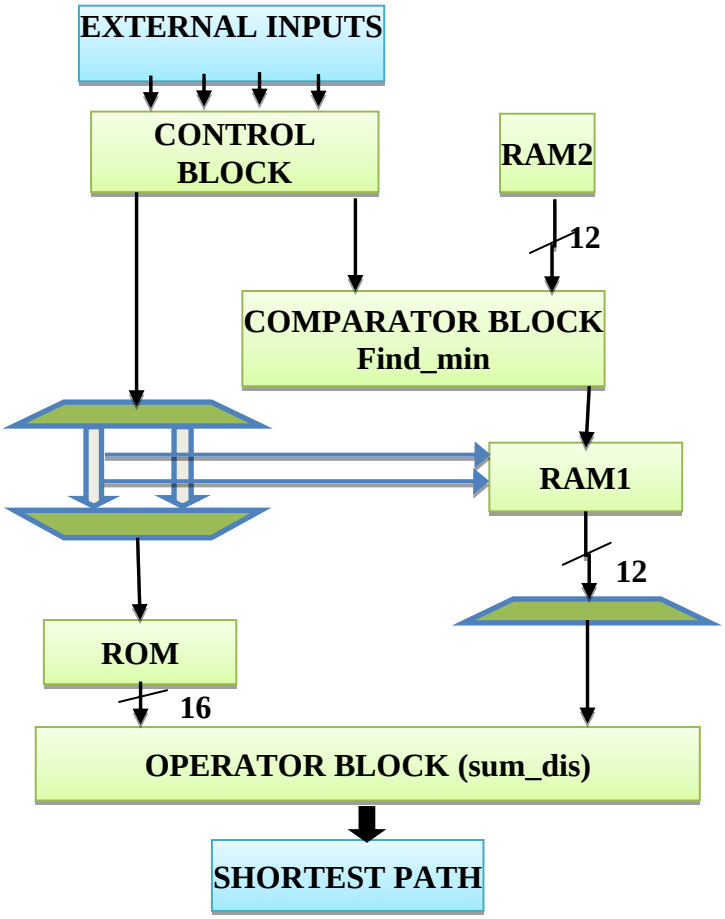


Figure 5.5 : Schéma de l'algorithme de Dijkstra [96]

5.4.4. Fonctionnement de l'architecture

Dans la figure 5.5 qui présente l'architecture proposée pour l'implémenter sur FPGA, nous n'avons pas séparé les bus d'adresses et de données pour le besoin de clarté du schéma.

Nous avons subdivisé l'architecture en blocs de descriptions qui sont : un bloc de commande, un bloc comparateur, un bloc opérateur et un bloc mémoire qui contient deux mémoires vives et une mémoire morte ROM.

A l'état initial, l'architecture prend en entrée la base de données topologique (la description de notre réseau) qui est une matrice d'ordre 2 dont les indices sont des nœuds et les éléments sont les coûts de lien entre ces nœuds, cette matrice sera stockée dans la ROM, en effet, elle est initialisée par la valeur de coût de liens entre les différents routeurs de notre réseau et la RAM1 est initialisée à 0 pour ''le nœud source R1'' et à l'infini pour le reste des nœuds. La RAM2 contient la liste de tous les nœuds qui n'ont pas encore été traités par le programme.

Le bloc de commande lit la liste des nœuds non traités dans la RAM2 et les transmet avec le nœud de traitement courant aux blocs RAM1 et ROM. Cette transmission est faite sous forme d'adresse, le bloc RAM1 recevant ces adresses et envoie les indices des nœuds qui lui sont

transmis au bloc opérateur, de même la ROM envoie, le coût de leurs liens au bloc opérateur. Le bloc opérateur réalise le calcul attendu, effectue ensuite une mise à jour des indices des nœuds du réseau contenu dans la RAM1, et bloque enfin l'espace contenant les données concernant le nœud traité (qui est ici 'R1' à la première boucle du programme).

Le bloc comparateur compare ensuite toutes les valeurs des indices des nœuds qui lui sont visibles, et renvoie le nœud d'indice minimal au bloc de commande. Celui-ci effectue une mise à jour des listes des nœuds dans la RAM2 en effaçant de cette liste le nœud d'indice minimal qui est maintenant devenu le nœud traité, et le cycle recommence.

Le programme s'arrête lorsque la RAM2 est vide.

Lorsque le programme est terminé, il y a encore un petit programme permettant de débloquer les données de laRAM1. A partir de ces données, on peut générer un fichier contenant les différents chemins détaillés du nœud sur lequel se trouve le paquet de données vers tous les autres nœuds du réseau physique (c'est en fait la table de routage).

En parallèle à ce travail principal qui consiste à implémenter l'algorithme de DIJKSTRA sur une carte FPGA, nous avons réalisé une application qui implémente le même algorithme de recherche du plus court chemin dans un processeur ordinaire. Nous avons utilisé l'outil Visual Studio 2008 et le langage C #. Nous avons mis en œuvre avec les classes que nous avons expliqué avant, en fait cette application était implémentée sur un micro-ordinateur portable avec un processeur Intel i7 et une RAM de 8GO pour extraire le temps d'exécution et au même temps illustrer le fonctionnement de cette architecture.

L'application nous permet d'insérer un nombre important de routeurs. Avec un simple clic sur l'interface de l'application, un routeur apparait sous le nom Ri suivant l'ordre d'application, chaque nœud a une étiquette et une paire de coordonnées (X, Y). Nous devons définir la liaison de chaque routeur avec les autres. Pour établir les liens entre les différents routeurs, on clique sur chacun des routeurs aux extrémités des liens. Une ligne noire montrant la liaison entre ces routeurs sera tracée et le coût de lien apparait automatiquement.

Avec l'ensemble des routeurs et les connexions de liens définis, nous pouvons calculer le chemin le plus court en choisissant le routeur d'origine, le routeur de destination et en cliquant sur le bouton ''*trouver le chemin*''. Dans ce cas, le coût est un facteur multiplicateur qui sera utilisé pour ajouter une sorte d'heuristique à la solution. La distance entre les routeurs est la mesure pour les calculs de plus court chemin. La distance est calculée automatiquement en cherchant la distance entre les deux coordonnées cartésiennes. Une ligne verte sera tracée pour montrer le plus court chemin entre les nœuds.

L'application nous donne aussi le temps d'exécution et le coût total du chemin trouvé et aussi le type du moteur d'exécution de l'algorithme dans le réseau proposé dans une barre en bas de l'interface de l'application.

On peut enregistrer et charger la carte dans un fichier texte. Ce fichier a un format très simple et il peut être modifié par un simple éditeur de texte comme Bloc-notes.

Dans l'exemple suivant nous proposons un réseau de 50 routeurs connectés par 82 arcs comme nous montre la figure 5.6.

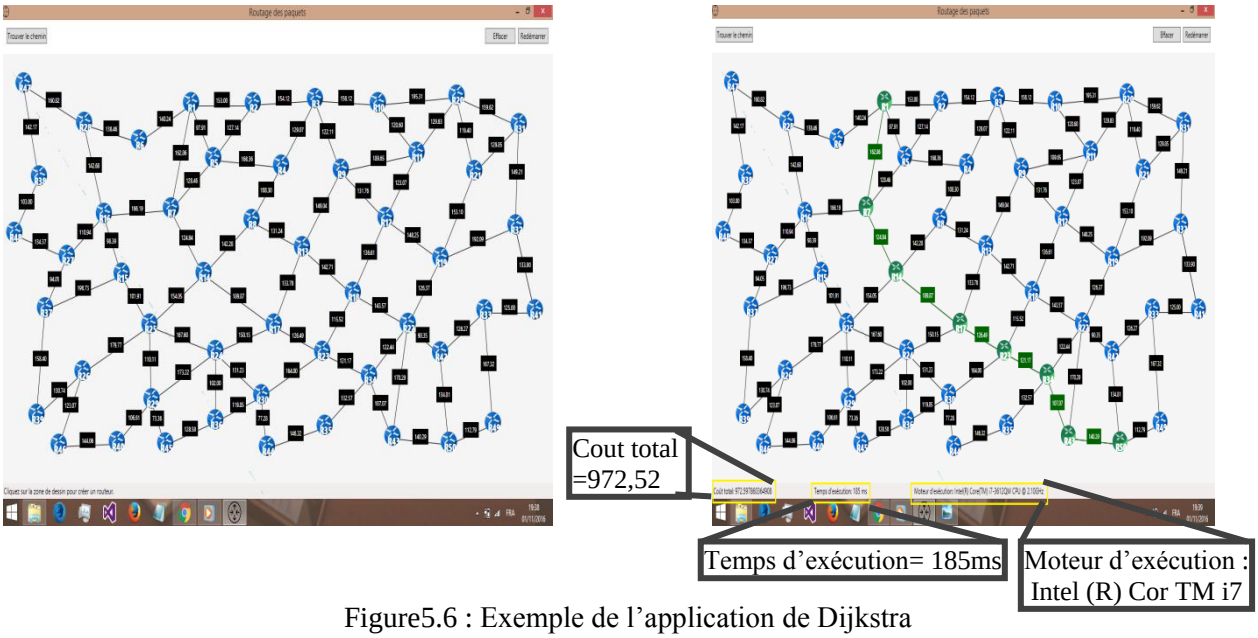


Figure5.6 : Exemple de l'application de Dijkstra

Pour trouver le chemin le plus court entre n'importe lequel des deux routeurs, il suffit de cliquer sur le bouton ''trouver le chemin'' puis on sélectionne le routeur de départ « R1 » et celui d'arrivée « R50 », le chemin le plus court sera tracé par une couleur différente (vert) R1-R7-R14-R17-R23-R34-R43-R50. En bas de l'écran, on trouve le coût total =972,52, le temps d'exécution= 185ms et le moteur d'exécution qui est un processeur Intel i7 comme nous montre la figure 5.6

La figure 5.7 présente un exemple de notre application sur les réseaux importants où on a utilisé un réseau de 50 routeurs et nous avons appliqué la technique des zones et les routeurs désignés DR (routeur en vert)

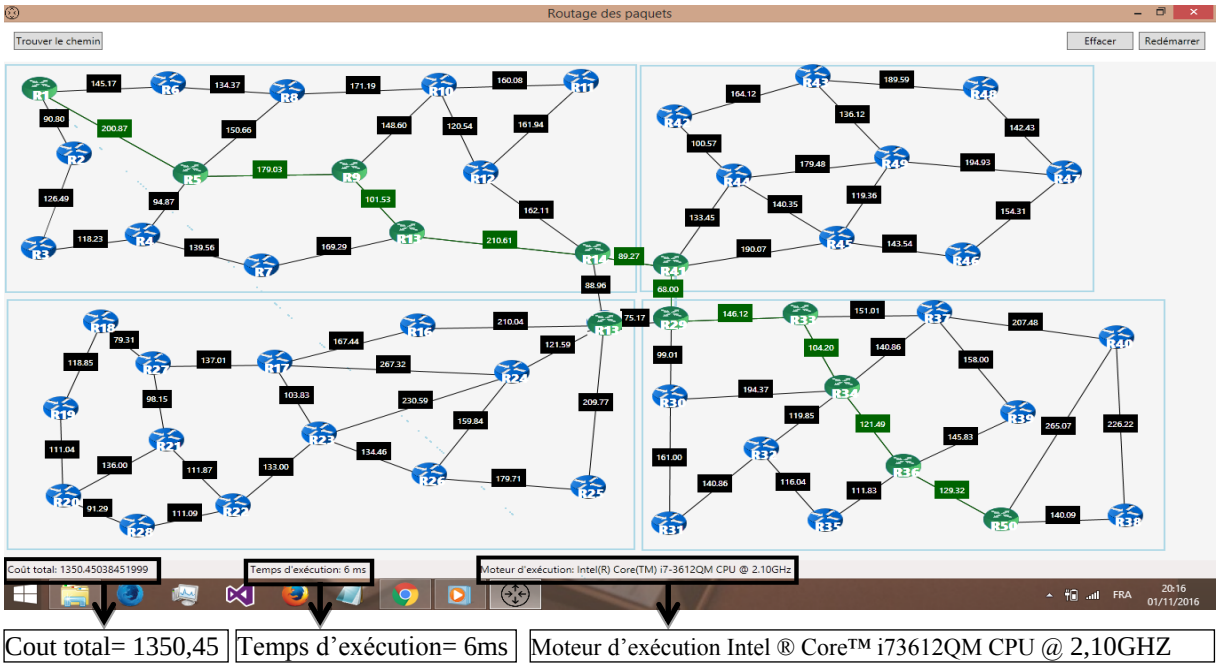


Figure 5.7 : exemple de l'application sur les réseaux importants

5.5. Résultats de l'implémentation de notre architecture sur FPGA

5.5.1. Description des blocs et implémentation sur FPGA

5.5.1.1. Bloc mémoire : contient une mémoire morte (ROM) et deux mémoires vives RAM1 et RAM2)

a) La ROM

La ROM est un package que nous allons définir dans la librairie *work* et ensuite nous allons juste l'appeler dans le programme principal. Les données de la ROM sont des vecteurs de 16 bits parce que les 4 premiers bits codent une extrémité du lien, les 4 suivants codent l'autre extrémité et les 8 derniers sont pour représenter la valeur du coût du lien entre ces deux extrémités. Par conséquent son bus de données aura 16 lignes. Son bus d'adresse contient 5 lignes pour pouvoir adresser tous les couples de nœuds.

Résultat de synthèse :

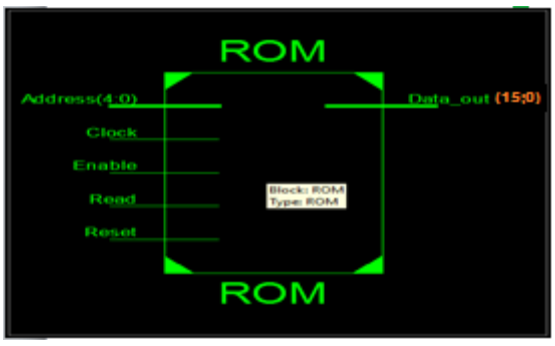


Figure 5. 8 : vue externe(RTL) du bloc ROM

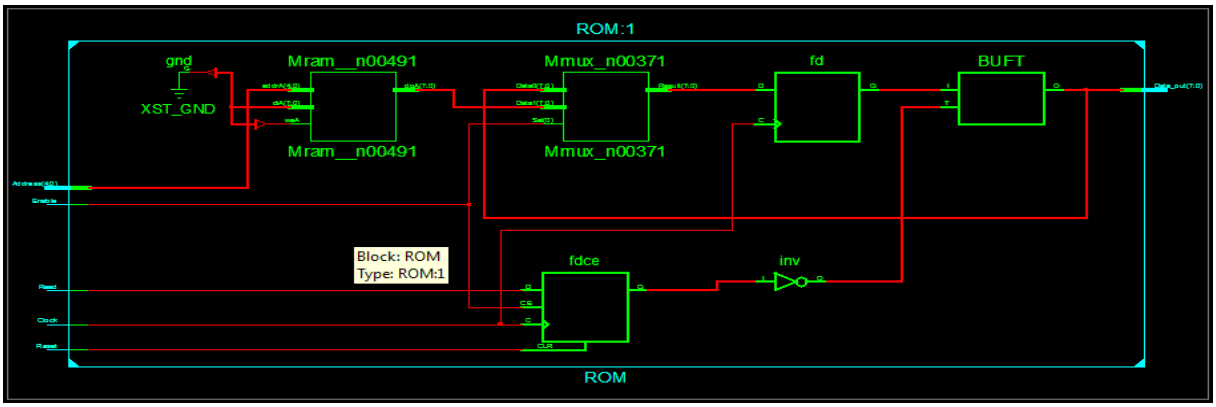


Figure 5.9 : vue interne (RTL) du bloc ROM

Le type mem_data définit un tableau (ARRAY) de 16 cases de largeur 8 bits. La constante data qui représente le contenu de la mémoire est initialisée aux lignes 10 à 26. Nous avons utilisé la fonction de conversion définie dans le package std_logic_unsigned, CONV_INTEGER. En effet, l'index du tableau data doit être un nombre entier alors que le signal data est de type std_logic_vector. La conversion std_logic_vector vers les entiers est obligatoire. La figure suivante montre le fonctionnement de la mémoire :

Simulation :

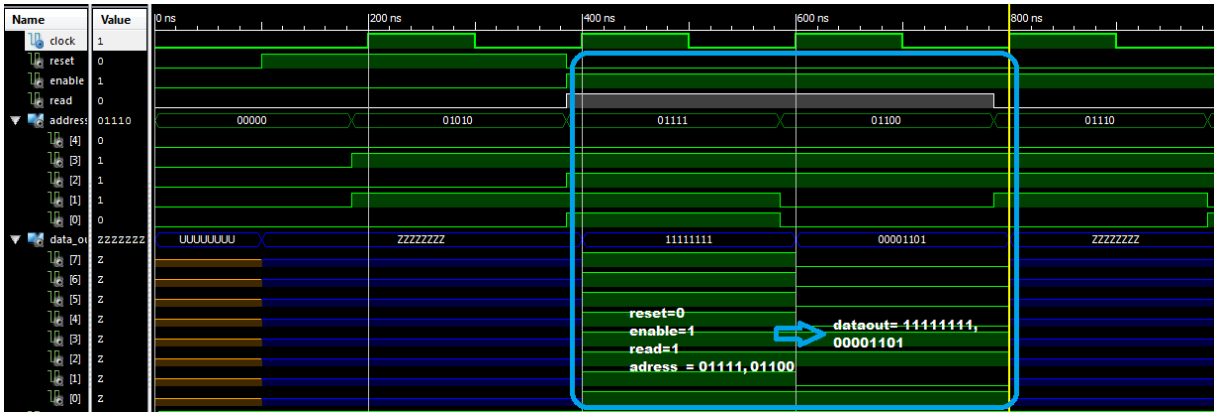


Figure 5.10 : résultats de simulation de bloc ROM

b) La RAM1

La RAM1 contient la valeur des indices des nœuds du réseau, son bus de données est constitué de 8 lignes dont les 3 premières sont pour coder le nœud et les 5 autres pour l'indice de ce nœud. Son bus d'adresse possède 6 lignes pour représenter tous les nœuds du réseau.

Résultats de synthèse :

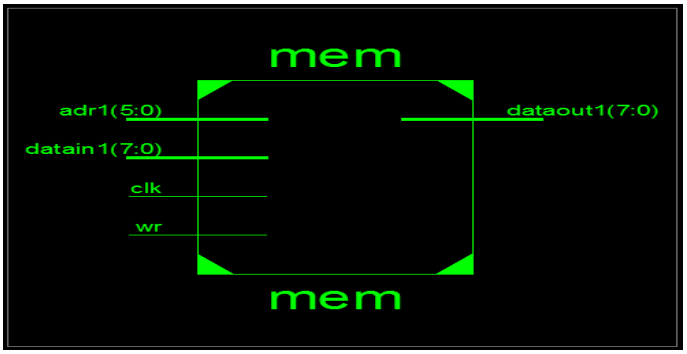


Figure 5.11 : vue externe (RTL) du bloc RAM1

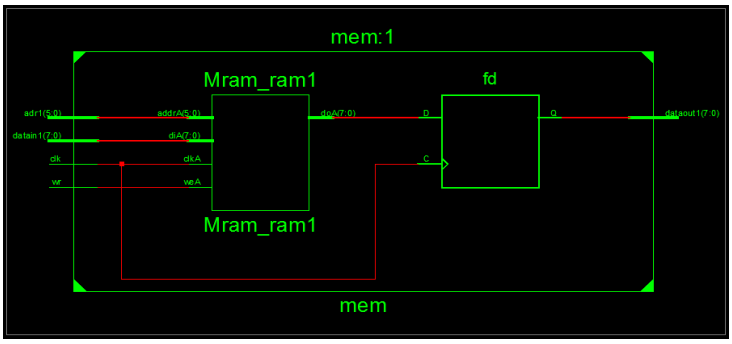


Figure 5. 12 : vue interne (RTL) du bloc RAM1

Simulation :

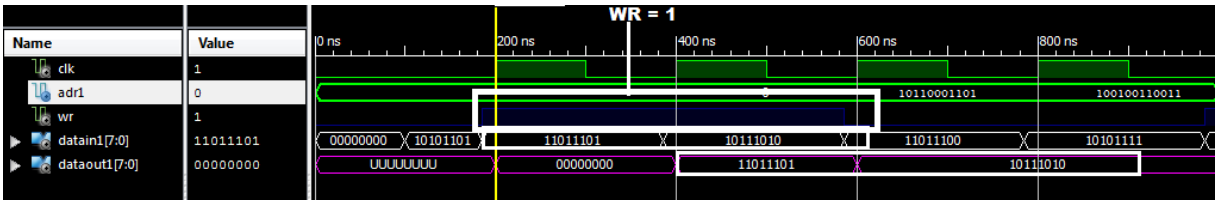


Figure 5.13 : résultats de simulation du bloc RAM1

5.5.1.2. Bloc opérateur (sum_dis) :

Le bloc **sum_dis** permet de faire la mise à jour des différents coûts de chaque nœud. On peut concevoir le bloc opérateur comme un processeur à part entière et la RAM1 utilisée est l'un de ses registres internes

Résultats de synthèse :

Figure 5.14 : vue externe (RTL) du bloc operateur

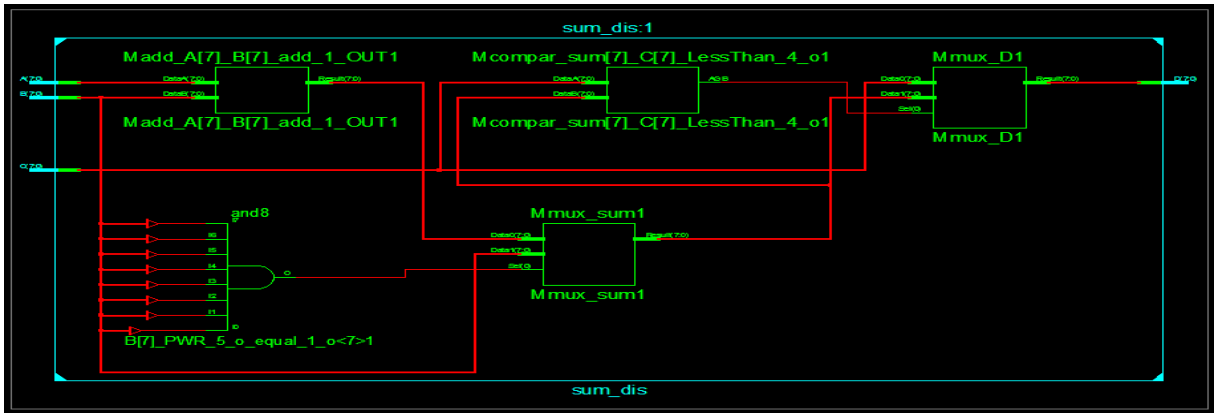


Figure 5. 15 : vue interne (RTL) du bloc Opérateur.

Simulation :

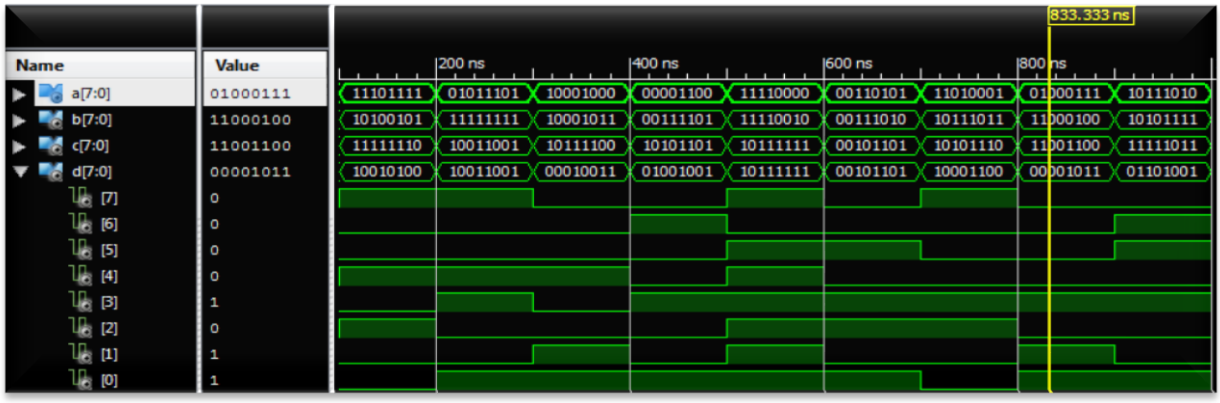


Figure 5. 16 : résultats de simulation de bloc sum_dis

5.5.1.3. Bloc comparateur (find_min) :

Les comparateurs sont des familles de circuit arithmétique. Ils effectuent sur N bits, les opérations d'égalité, de supériorité ou d'infériorité. Le circuit suivant est un bloc comparateur de trois sorties où chaque sortie est le résultat d'une comparaison de deux entrées, donc le bloc **find_min**, compare les différents coûts de chaque nœud et retient le plus petit.

Figure 5. 17 : vue externe (RTL) du bloc Comparateur.

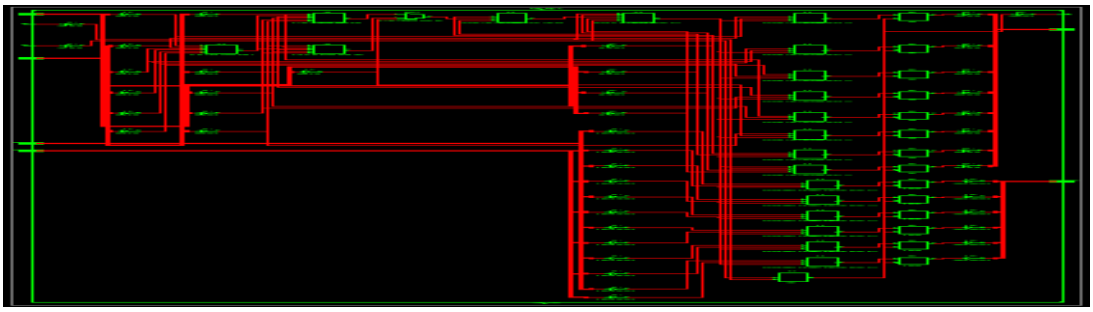


Figure 5. 18 : vue interne (RTL) du bloc comparateur

Le bloc **find_min** est activé en simulation et le synthétiseur génère des portes logiques pour ce design. En effet, C représente la valeur min entre A et B qui sont codés sur 8 bits, l'index pour extraire la valeur min entre index1 et index2 est codé sur 6 bits, et le flag représente toujours la valeur minimum entre flag1 et flag2.

Simulation :

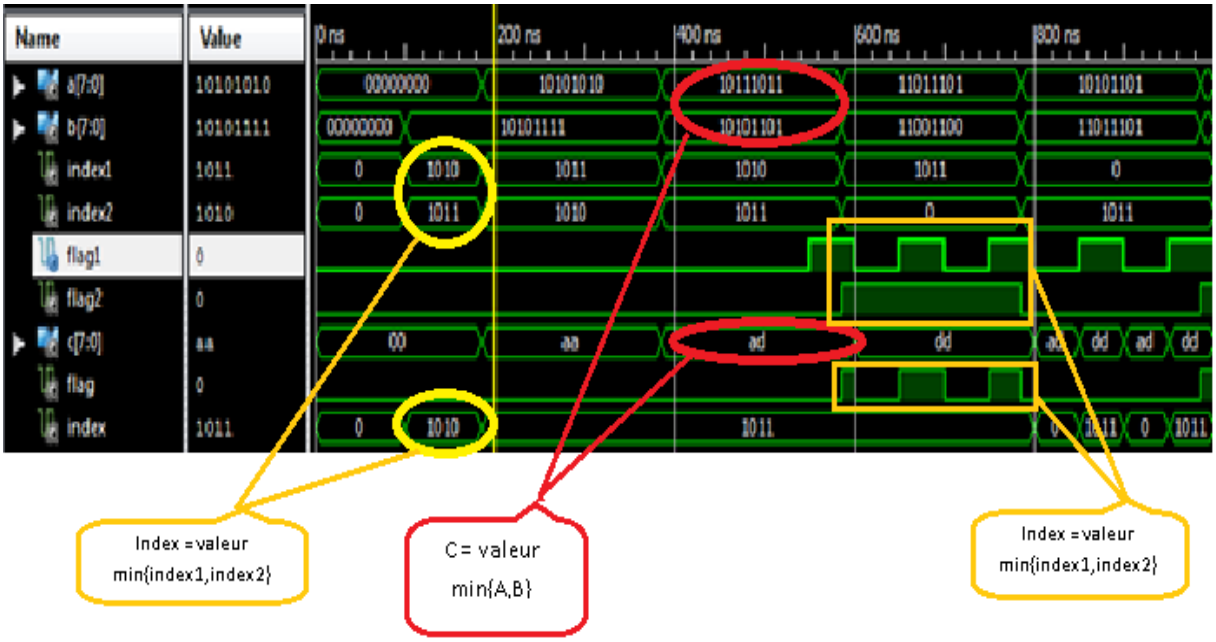


Figure 5. 19 : résultats de la simulation du bloc comparateur

5.5.1.4. Synthèse du bloc général de l'algorithme de DIJKSTRA

Rappelons que l'objectif principal de notre implémentation est de voir le temps d'exécution de l'algorithme de DIJKSTRA sur un routeur à base de processeur FPGA. Les figures (5.20 et 5.21) montrent successivement une vue externe et une vue interne de notre circuit FPGA.

La figure 5.22 est un aperçu de la simulation d'un routeur FPGA. A l'instant t = 0 ns, débute une décision de routage d'un paquet de données vers sa destination. Le transfert des données ne se fait qu'après l'étape passée par les blocs qu'on a expliqués précédemment (commande, find_min, sum_dis…)

Résultats de la synthèse

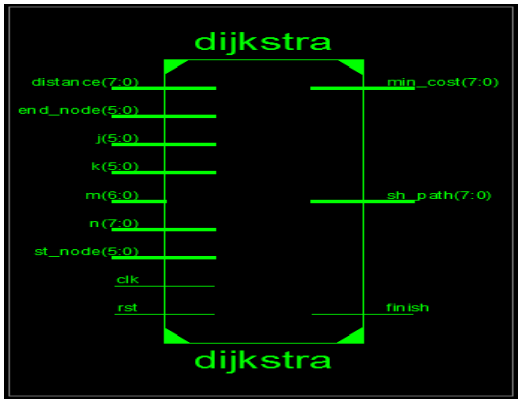


Figure5. 20 : vue externe du circuit

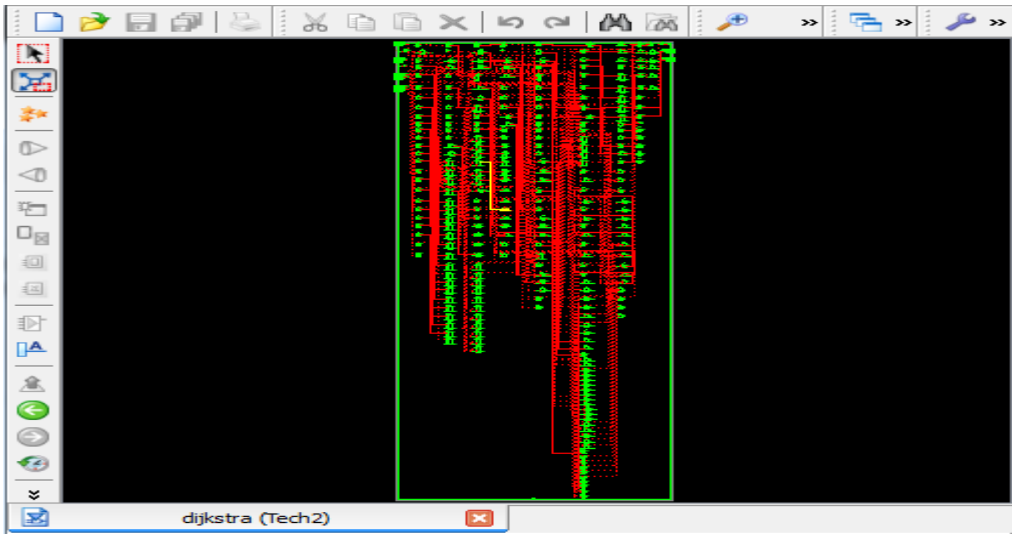


Figure 5. 21 : vue interne schéma RTL de l'implémentation de l'algorithme sur Virtex 7 [96]

Simulation

Les processeurs désignés à base de FPGA, pour n = 10, 20, 30, 40, et 50, sont conçus en VHDL selon l'architecture proposée pour l'implémenter sur puce FPGA, Xilinx Vertix-7 (XC7V2000T). Elles sont testées sur diverses topologies réseau et avec différentes complexités de coût et de densité de connexion. La Figure 5.22 montre la simulation de l'implémentation de notre architecture sur FPGA et la principale entrée / sortie et d'autres signaux pour les routeurs DR à base de FPGA. La simulation est réalisée par Xilinx ISIM simulateur [96].

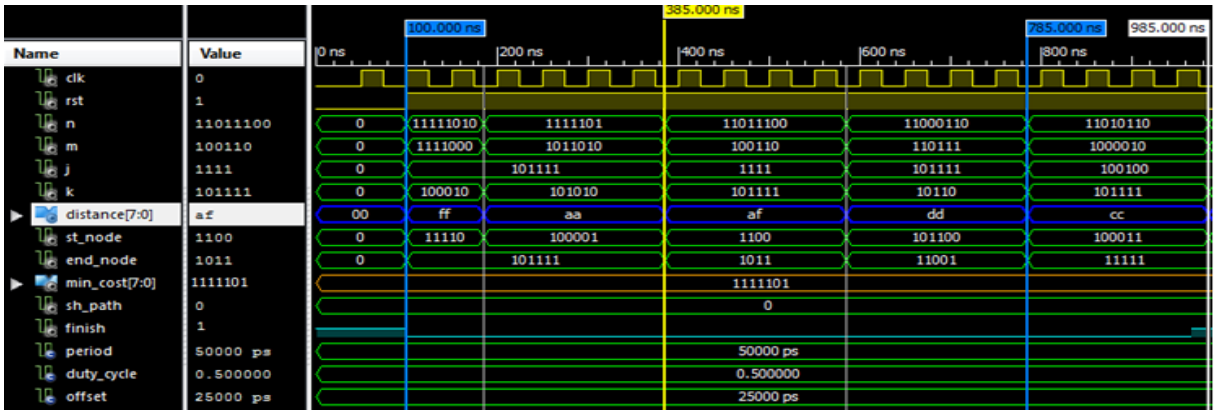


Figure 5. 22 : Résultats de simulation de l'implémentation de l'algorithme sur FPGA [96]

L'entrée principale est une matrice de coût de lien qui représente la topologie du réseau utilisé (présentée sur la figure 5.7) avec le routeur R1, (n (1,1)) est le nœud de départ, notez que les coûts de liaison sont représentés ici en nombres hexadécimaux et le coût de lien NN représente l'infini. Le nombre d'horloges et le nombre d'itérations consommées par le processeur DR sont clairs à noter : Le chargement et le verrouillage de la matrice des coûts prennent 385.00 ns. Le processus de calcul du bloc opérateur dure quatre itérations pour compléter l'algorithme et son temps écoulé est 785,00 ns. On peut remarquer que le vecteur de distances est mis à jour après chaque itération, des cycles d'horloge supplémentaire sont nécessaires pour extraire les informations de la table de routage et le temps de chargement-calcul-extraction total sera de 900,00 ns.

5.5.2. Le rapport de consommation des ressources :

L'environnement ISE fournit un rapport de synthèse sous forme de tableaux contenant les informations utiles liées au design que voici :

DIJKSTRA Project Status (08/24/2016 - 23:02:29)			
Project File:	projectdijkstra.xise	**Parser Errors:**	No Errors
Module Name:	DIJKSTRA	**Implementation State:**	Synthesized
Target Device:	xc7v2000t-2flg1925	• **Errors:**	No Errors
Product Version:	ISE 14.2	• **Warnings:**	33 Warnings (33 new)
Design Goal:	Balanced	• **Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	
Environment:	System Settings	• **Final Timing Score:**	

Tableau 5. 1 : l'état du projet de DIJKSTRA

Le tableau 5. 1 illustre l'état du projet, il ne donne pas d'erreurs, 33 Warnings sans influence sur le bon fonctionnement de l'algorithme.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	6558	607200	1%
Number of Slice LUTs	22690	303600	7%
Number of fully used LUT-FF pairs	2916	23332	12%
Number of bonded IOBs	59	700	8%
Number of BUFG/BUFGCTRLs	3	32	10%

Tableau 5. 2 : ressources logiques utilisées à l'implémentation physique.

D'après le tableau 5.3 nous pouvons dire que la consommation d'énergie par le FPGA croit avec l'importance du réseau ce qui est tout à fait logique car il faut davantage de ressource mémoire et de ressource opératoire.

Environment (nodes and arcs)	**Frequency (MHz)**	**Power consumed (mW)**
30 routeurs (et 54 arcs)	92	1875
50 routeurs (et 72 arcs)	97.7	3142

Tableau 5. 3 : Consommation d'énergie

5.6. Résultats de comparaisons entre un FPGA et un processeur ordinaire

On a exécuté le programme de Dijkstra codé en VHDL sur l'outil ISE 14.2 de Xlinx, le même algorithme codé en C # est exécuté sur un micro-processeur ordinaire. Un exemple commun d'architecture de réseau a été testé séparément sur chacun des deux processeurs, les résultats sont consignés dans les figures ci-dessous :

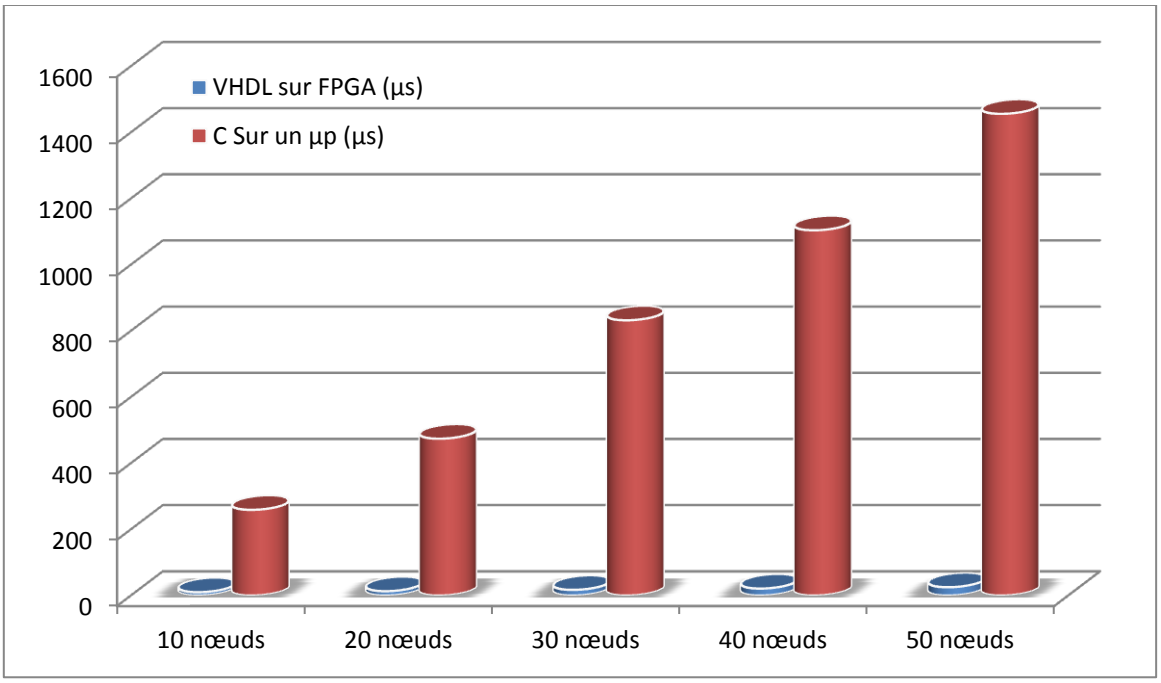


Figure5. 23 : Temps d'exécutions de l'algorithme de DIJKSTRA

Le point que nous ferons ressortir de la figure 5.23 est évidemment sur les différents temps d'exécution de l'algorithme. Tout d'abord sur le FPGA comme sur le microprocesseur, le temps d'exécution de l'algorithme croit avec l'importance du réseau, ce qui est évidemment tout à fait logique.

La figure 5.23 nous montre aussi combien le FPGA exécute plus vite qu'un microprocesseur conventionnel. Cela est dû à des raisons essentielles :

✓ Tout d'abord la différence est au niveau de l'architecture interne du FPGA qui est conçue pour effectuer des tâches dédiées c'est-à-dire bien spécifiques en sorte que pendant son fonctionnement, il n'exécute que la tâche qui lui a été assignée, c'est ce qui lui confère sa rapidité. Le microprocesseur conventionnel quant à lui effectue plusieurs tâches à la fois ce qui cause son retard.

✓ Le FPGA a également cet avantage que sa programmation allie la souplesse du software à la rapidité du hardware, alors que celle du microprocesseur est réalisée de manière soft uniquement et ne bénéficie donc pas de la rapidité du hard que possède le FPGA.

✓ Les instructions multiples sur les variables sont exécutées de façon prioritaire sur le FPGA, les opérations arithmétiques multiples, telles que les comparaisons, sont exécutées en parallèle.

La figure suivante représente des rapports temporels, nous montre combien le FPGA exécute de loin plus vite qu'un microprocesseur conventionnel.

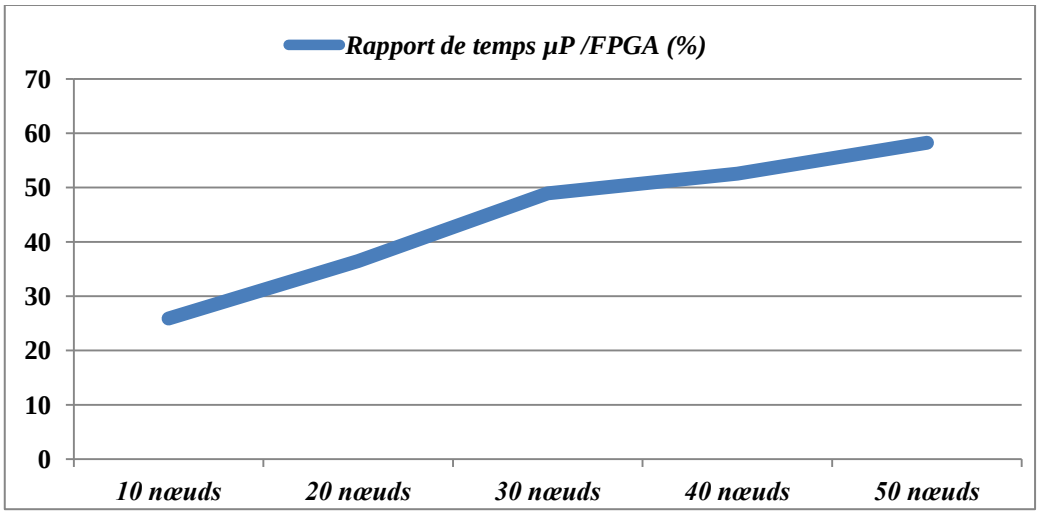


Figure 5.24 : Rapport de temps µP /FPGA

5.7. Transfert du résultat solution du programme vers un support physique FPGA

Cette étape est la dernière dans le processus de conception en général et en particulier de notre projet. Elle va permettre de charger le fichier de description VHDL sur un support FPGA (Migration de la solution vers la cible hardware) choisi au préalable. Pour notre cas, une interface fournie par XLINX pour la configuration du FPGA par un fichier en format BIT. Le transfert de la solution finale peut être assuré par un câble JTAG (joint Test Action Group) qui relie le micro-ordinateur (PC) et la carte FPGA.

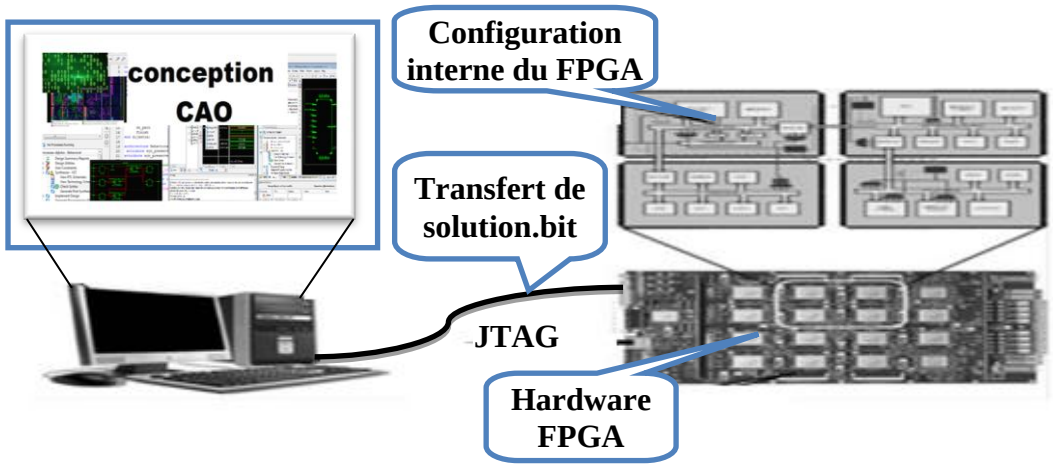


Figure 5. 25 : configuration du FPGA par un câble JTAG

5.8. Conclusion

Ce dernier volet de cette étude nous permet de conclure que les résultats obtenus démontrent la justesse des modèles retenus sachant que cette méthode de conception permet de découvrir les problèmes de vérification puis de réduire le facteur d'erreur humaine et les risques d'interruption du système. La conception et la simulation du circuit numérique menée par

l'appui d'outils informatiques spécialisés et l'utilisation du langage VHDL comme outil de description pour représenter le comportement et l'architecture du dispositif numérique nous ont permis d'obtenir un certain niveau de réutilisabilité des différents blocs de l'architecture mais toute la difficulté est de savoir ce qui est synthétisable ou non ainsi la difficulté de construire le « Test-Bench » afin de simuler la description.

Notre architecture est acceptable en termes de vitesse de fonctionnement. En outre, les ressources utilisées sont relativement faibles. Enfin, tous les résultats obtenus révèlent l'efficacité de notre conception.

CONCLUSION GENERALE ET PERSPECTIVES

Conclusion générale et perspectives

L'augmentation continue de la capacité d'intégration dans les circuits reconfigurables [97], et la complexité croissante des applications communicantes [98], nous ont motivé à utiliser des cartes FPGAs pour accélérer le routage de l'information dans les réseaux de télécommunication. En effet, la généralisation des réseaux de télécommunication et l'évolution technologique est généralement motivée par la conception des systèmes embarqués.

Le domaine de la conception et de l'implémentation sur des circuits numériques à architecture reconfigurable comme les FPGAs est complexe car il nécessite une maitrise des technologies relatives aux FPGAs et aussi une très bonne connaissance des applications et de leurs environnements [99].

Après la présentation des différents langages et niveaux de description du matériel ainsi que les outils de développement, nous avons choisi le VHDL comme langage de programmation à cause de sa disponibilité et son efficacité pour la description de notre architecture. Nous avons choisi les outils de conception de Xilinx par rapport à ceux d'ALTERA du fait que les premiers existent dans notre laboratoire, leurs flots de conception sont simples et rapides mais aussi du fait que l'utilisation des plateformes multi-composants de la même compagnie (Xilinx) y figurent. En effet, pour la validation de notre implémentation, nous avons utilisé le circuit FPGA Virtex7(XC7V2000T).

Nous avons étudié en détails le routage dans les réseaux de télécommunications et nous avons conclu que les routeurs actuels utilisés dans les réseaux informatiques ont des difficultés croissantes avec les qualités et les exigences des services devant être assuré par les réseaux.

Dans le cadre de ce projet nous avons développé une méthodologie de conception et d'implémentation d'une architecture de l'algorithme de Dijkstra pour un système de routage. La démarche théorique suivie est illustrée comme suit :

- ✓ Spécification du cahier de charge avec une étude théorique
- ✓ Conception architecturale détaillée
- ✓ Test et intégration
- ✓ Validation.

Durant le processus de conception, nous avons constaté que malgré les progrès réalisés dans le domaine des architectures reconfigurables, les outils de programmation n'ont pas atteint leurs maturités et que les algorithmes de programmation matérielle s'éloignent

significativement des algorithmes logiciels. Nous avons démontré dans notre recherche, la possibilité d'implémentation matérielle de l'algorithme de Dijkstra. Sachant que le développement des systèmes électroniques soit encore très dépendant de la technologie d'implantation, nos résultats attestent que les FPGAs constituent une alternative sérieuse aux processeurs ordinaires. Donc, le FPGA est un moyen d'améliorer les performances de routage avec un gain économique en énergie et en temps d'exécution.

Dans cette thèse, une architecture du processeur à base de FPGA a été proposée pour accélérer le processus afin de trouver le plus court chemin et construire la table de routage des réseaux OSPF. L'architecture de conception de notre processeur proposée est basée sur l'algorithme de Dijkstra. Le fonctionnement du processus est présenté sur trois phases : Le chargement de la matrice des coûts, le calcul du plus court chemin informatique, et l'extraction des informations de la table de routage. Nous avons remarqué que les temps requis pour le chargement de la matrice de coût et l'extraction des informations de la table de routage sont plus lents pour les grandes tailles de réseau mais on peut les réduire moyennant ces techniques. Par exemple, pour n = 50, le temps de chargement et d'extraction est presque de 42,77% du temps total de calcul du processeur. Pour la taille de réseau de 100 routeurs nous avons atteint un rapport de temps égal à 76.77%.

Nous avons remarqué aussi que la consommation des ressources FPGA est également augmentée de façon exponentielle avec l'augmentation de la taille du réseau. Pour N = 50, le processeur occupe 38% des éléments logiques disponibles sur la carte FPGA Xilinx Vertix-7 (XC7V2000T). Il est clair que si nous continuons à augmenter la capacité du processeur pour des réseaux à grande échelle, par exemple, un réseau de 250 nœuds, avec la même stratégie, les ressources logiques nécessaires dépasseront la capacité totale de la puce FPGA. Pour cette raison, nous avons segmenté le réseau en zones et utilisé des routeurs DRs qui sont à base de FPGA, en effet, pour un réseau de 500 routeurs, on peut trouver 25 zones de 20 routeurs donc on a besoin de 25 routeurs désignés (DR) ou plus, actuellement nous limitons le travail à l'implémentation de notre architecture sur un processeur à base de FPGA capable de gérer les réseaux de 50 routeurs désignés.

Finalement, l'architecture proposée dans ce travail peut être avantageusement améliorée et facilement étendue sur d'autres types de protocoles réseaux et les chercheurs du domaine vont donc devoir relever les défis des problèmes liés à la technologie du futur.

Dans le cadre des systèmes à hautes performances, ce travail présente une contribution à la conception et à l'implantation d'un algorithme de routage dans les réseaux de

télécommunications sur FPGA. Plusieurs travaux futurs vont compléter le travail déjà initié par l'élaboration de notre thèse à savoir :

- Améliorer et optimiser le model VHDL de notre architecture proposée, c'est à dire une réécriture du code VHDL de certains opérateurs de calcul dans le but d'accélérer le temps d'exécution ou la vitesse de fonctionnement de l'algorithme et minimiser les ressources matérielles nécessaires.
- L'adaptation dynamique du design par la reconfiguration dynamique des composants FPGA (Field Progrmmable Gate Array).
- L'amélioration dans la conception conjointe logicielle / matérielle qui est apparue comme une première réponse au co-design de la reconfiguration dynamique.

Il reste à chercher un moyen très efficace de combiner les avantages des deux approches de programmation. Dans ce cas, toute la difficulté consiste à répartir les tâches sur les différents processeurs et y définir les architectures respectives.

- Ajouter des blocs supplémentaires dans l'architecture et des contraintes pour pouvoir effectuer la mise à jour des différents modules séparément.
- Les aspects énergétiques des opérateurs seront l'objet de recherche poussée de ces problématiques.
- L'optimisation du prototype aussi bien en termes de densité de ressources logiques que de temps d'exécution.
- Différents points n'ont pas été réalisés dans cette thèse, on espère que des travaux futurs se feront dans le but d'améliorer ces architectures où le parallélisme nous semble être le plus efficace pour des composants toujours plus rapides.

REFERENCES

Références bibliographique

[1] P. S. Obwang, "Le Routage à Qualité de Service dans les Réseaux Mobiles Ad Hoc", Thèse de doctorat, université de Valenciennes et du Hainaut-Cambresis, 2010.

[2] D. G. Barrera, "A differentiated quality of service oriented multimedia multicast protocol", Thèse de doctorat, université de Toulouse, INPT, 2005.

[3] J. Chanet, "Algorithme de routage coopératif à qualité de service pour des réseaux ad hoc agri-environnement ", Thèse de doctorat, université de Clermont-Ferrand 2, 2007.

[4] M. Turki, "Techniques de multiplexage pour un système d'émulation et de prototypage rapide à base de FPGA, " Thèse de doctorat, université de Paris-6 (France) en cotutelle avec l'université de Sfax (Tunisie), 2014.

[5] M. S. Azzaz, "Implantation paramétrable d'un nouvel algorithme de cryptage symétrique basé Chaos par inclusion au sein d'une architecture reconfigurable de type FPGA", Thèse de doctorat, université de Université de Lorraine (France) en cotutelle avec École Militaire Polytechnique (Alger), 2012.

[6] CISCO, "Guide de conception OSPF", http://www.cisco.com/cisco/web/support/CA/fr/109/1091/1091859_1.pdf

[7] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ibmsna.htm#xtocid8.

[8] http://www.ISO.org.

[9] K. Iso, T. Watanabe, "Speaker-independent word recognition using a neural prediction model" In ICASSP, pp 441-444, 1990.

[10] Technologies de l'information " Interconnexion de systèmes ouverts (OSI) ", Modèle de référence de base : Le modèle de base ISO/IEC 7498-1 :1994.

[11] Ph. Latu, "Modélisation en couche des protocoles réseau", Documentation Libre GNU, http://www.inetdoc.net/pdf/modelisations.pdf, cité le 30/04/2016

[12] P. ROLIN, G. MARTINEAU, L. TOUTAIN, A. LEROY, "Les réseaux, principes fondamentaux", Hermes, 574p, décembre 1996.

[13] G. Pujolle, "Les Réseaux", Eyrolles 3eme édition, 2002.

[14] P. Stéphane, "Communications dans les réseaux d'interconnexion", thèse de doctorat de l'université de NICE, 1996.

[15] F. Castel, "Les télécommunications, dirigé par", éditions Berger-Levrault International, 1993.

[16] V. Jacobson, "Congestion Avoidance of Network Traffic", Computer Communication Review, vol. 18, no. 4, pp. 314-329, 1988.

[17] V. Jacobson, K. Nichols, "An Expedited Forwarding PHB", RFC 2598, Juin 1999.

[18] K. Steenhaut, M. Fakir, A. Nowé et E. Dirkx, "Reinforcement learning for revenue optimization in ATM networks: a case study based on simulation". Proceedings of the IASTED International Conference, Modelling and Simulation, Pittsburgh Pennsylvanian USA, pp 52-56, 2015.

[19] M. Lottor, "Internet Growth (1981-1991)", RFC 1296 January 1992.

[20] J. Hawkinson T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", RFC 1930 March 1996.

[21] J. Moy, "OSPF specification", RFC 1131 1989.

[22] R. Callon, "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments", RFC 1195, December 1990.

[23] D.L. Mills, "Exterior Gateway Protocol Formal Specification", RFC 904 April 1984.

[24] K. Lougheed, Y. Rekhter, "A Border Gateway Protocol 3 (BGP-3)", RFC 1267, October 1991.

[25] Cisco Networking Academy http://balma.afpamp.org/RandS/RandS-2/index.html,

[26] R. LEGRAND et A. VAUCAMPS, "Les réseaux avec Cisco", 2ième édition, Édition : ENI - 471 pages, 2015, ISBN10 : 2746092638 - ISBN13 : 9782746092631.

[27] J. DORDOIGNE, "Réseaux informatiques", Édition : ENI, 603 pages, 6ieme édition, 2015, ISBN10 : 2746093928 - ISBN13 : 9782746093928

[28] D. Paret, "Réseaux multiplexés pour systèmes embarqués", Édition : Dunod, 2e édition, 440 pages, 2012, ISBN10 : 2100052675 - ISBN13 : 9782100582891

[29] Ch. HUITEMA, "Le routage dans l'Internet, EYROLLES", 418 p, 1994, ISBN: 2-212-08902-3.

[30] E. Crawley, R. Nair, B. Rajagopalan et H. Sandick, "A Framework for QoS-based Routing in the Internet", RFC2386, IETF, August 1998.

[31] Z. Sanchez, Salkewicz, Crawley, "Quality of Service Extensions to OSPF or Quality Of Service Path First Routing (QOSPF)", Internet Draft draft-zhangqosospf-01.txt, September 1997.

[32] S. HOCEINI, "Techniques d'Apprentissage par Renforcement pour le Routage Adaptatif dans les Réseaux de Télécommunication à Trafic Irrégulier", thèse de doctorat de l'université paris val de marne, 2004.

[33] S. Chen and K. Nahrstedt, "On finding multi-constrained paths" ICC'98, Atlanta, Georgia, June 7-11, 1998 pages.874-879.

[34] Jaffe J. M., "Algorithms for Finding Paths with Multiple Constraints. Networks", Networks an international journal, 1984. vol. 14: p. 95– 116.

[35] Ch. Glacet. "Algorithmes de routage : de la réduction des coûts de communication à la dynamique", thèse de doctorat de l'université Sciences et Technologies Bordeaux I, 2013.

[36] A. HAJAMI, "Sécurité du routage dans les réseaux sans fil spontanés : cas du protocole OLSR", thèse de doctorat de l'université Mohammed V- Rabat, 2011.

[37] O. Cogis, C. Robert, Théorie des graphes, Vuibert, 2003

[38] F. Droesbeke, M.Hallin, Lefevre Cl., Les graphes par l'exemple, Ellipses, 1987

[39] L. Seymour, Mathématiques discrètes, Série Schaum, McGraw-Hill, 1990

[40] M. Gondran, M. Minoux, Graphes et algorithmes, 2e édition, Eyrolles, 1986

[41] J.C Fournier, Théorie des graphes et applications, Lavoisier, 288 pages, 2007.

[42] S. Skiena, "Implementing Discrete Mathematics", Addison-Wesley, 1990 (http://www.combinatorica.com)

[43] E.W. Dijkstra " A short introduction to the art of programming ", 1971, contenant l'article original (1959) décrivant l'algorithme de Dijkstra (pages 67 à 73).

[44] J. Moy, "Using the Dijkstra algorithm, a tree is formed from this subset of the link state database", RFC2328 OSPF Version 2, 1998, consulté le 19 Avril 2016 : p. 161.

[45] R. David " An algorithm invented by Dijkstra known as shortest path first (SPF)", RFC1142: OSI IS-IS Intra-domain Routing Protocol, 1990, consulté le 30 mai 2016.

[46] P. Lacomme, "méthodes exactes et approchées pour l'optimisation des systèmes à moyens de transport". Mémoire pour l'obtention de l'Habilitation à Diriger des Recherches de l'Université Blaise Pascal (Clermont –Ferrand II), 2015.

[47] E.Q.V. Martins, M.M.B. Pascoal et J.L.E. Santos, "The K Shortest Paths Problem", Research Report, CISUC, 1998.

[48] A. Nketsa, "Circuits logiques programmables : mémoires, PLD, CPLD et FPGA", Éditeur : Ellipses, Collection : Technosup, 1998,

[49] C. TAVENIER, "Circuits logiques programmables". PARIS: DUNOD, 1996.

[50] J.WEBER et M.MEAUDRE, "Circuits numériques et synthèse logique, un outil VHDL", Edition Masson collection technologie, 2006.

[51] E. MESSERLI, "Manuel VHDL synthèse et simulation", Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (heig-vd), septembre 2007.

[52] J. DETREY, "Arithmétiques réelles sur FPGA virgule fixe, virgule flottante et système logarithmique", Thèse de doctorat de l'Ecole Normale Supérieure de Lyon, Janvier 2007.

[53] K.-J. Lee, T.-Y. Hsieh, C.-Y. Chang, Y.-T. Hong, and W.-C. Huang, "On-Chip SOC Test Platform Design Based on IEEE 1500 Standard", IEEE Trans. Very Large Scale Integr. Syst., vol. 18, no. 7, pp. 1134–1139, Jul. 2010.

[54] S. SNAIDERO, "Modélisation multidisciplinaire VHDL-AMS de systèmes complexes vers le prototypage virtuel", thèse de doctorat de l'université Louis Pasteur Strasbourg I, décembre 2004.

[55] D. GUIHAL, "Modélisation en langage VHDL-AMS des systèmes Pluridisciplinaires", thèse de doctorat de l'université de Toulouse 3, Mai 2007.

[56] Synopsys, "SaberHDL : language-Independant Mixed –Signal Multi-Technology Simulator", Synposys Inc, Etats-Unis d'Amérique 2003.

[57] D.SMITH, "HDL Chip Design: A pratical Guide for Designing, Synthesis & Simulating Asics &FPGAs using VHDL or Verilog", Doone Pubns, 2006.

[58] "Electrical Engineering Times", October 2008. [Online]. Available: http://www.eetimes.com/electrical-engineers/education-training/courses/4000134/Fundamentals-of-FPGAs. Consulté février 2012.

[59] D. DECKERS, "Composants programmables", cours en Electronique appliquée, Institut Supérieur Industriel de Mons Haute Ecole de la Communauté française en Hainaut Unité Electronique 2008-2009.

[60] N. Heidmann, T. Wiegand, S. Paul, "Architecture and FPGA-implementation of a high through put K+-Best detector". In: Design, Automation and Test in Europe, (2011).

[61] S. G. Dighe, M. T. Kanawade, " Field Programmable Gate Array Technique's ", International Journal of Computing and Technology, Volume 2, Issue 12, December 2015.

[62] P.MELET, "Conception et réalisation d'un circuit numérique spécifique à étalement de spectre pour un système multi capteurs en milieu clos", thèse de doctorat de l'université Paul Sabatier, Toulouse, Dec, 2001.

[63] C.GUILLEMINOT, L.ANDRIEUX, JJ.MERCIER, "A contribution to a virtual prototyping for a spread spectrum telecommunication system using VHDL-AMS", IEEE BMAS'05, SanJose, Californie, USA, 22-23, 2005.

[64] V. Betz and J.Rose, "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density", In International Symposium on Field Programmable Gate Arrays, 1999.

[65] XLINX, DS112 V3.1, http://www.xilinx.com/about/xcell-publications/xcell-journal.html, consulté novembre 2014.

[66]O.r. Guiller, "Intégration de capacités verticales débouchantes au sein d'un interposeur silicium", Thèse de doctorat de l'université GRENOBLE, avril 2015

[67] P. Dorsey. "Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency". Technical Report Xilinx White paper: Vertex-7 FPGAs, 2011.

[68] C. Maxfield, "New Xilinx virtex-7 2000t FPGA provides equivalent of 20 million asic gates", 2011.

[69] Virtex-7 (2014). 7 Series FPGAs Overview. Available from: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[70] C. Guex, E. Messerli, "Circuits programmables et langages de conception, une évolution en parallèle", Revue Vision, EIVD, 1998.

[71] IEEE Standard VHDL Language Reference Manual (VHDL-1993), IEEE 1076-1993.

[72] B. A. A. Antao, A. J. Brodersen, "Behavioral simulation for analog system design verification", IEEE custom integrated circuits conference, 1994.

[73] B. A. A. Antao, R. Saleh, "Simulation model transformations for analog hardware description languages", IEEE custom integrated circuits conference, 1995.

[74] D. Gauthey & E. Messerli, "Conception numérique: Description VHDL et synthèse", Revue Vision, EIVD, 2000.

[75] IEEE Standard Multivalue Logic System for VHDL Model Interoperability, IEEE Std 1164-1993

[76] J. WEBER, "Le langage VHDL". PARIS: DUNOD, 2000.

[77] VerilogA, Reference manual, Analog Extensions to Verilog HDL, Cadence, 1996.

[78] G. R. Smith, FPGA 101. New York: ELSEVIER, 2010.

[79] J. Dabrowski, "Functional-Level analog macro modeling with piecewise linear signals", proceeding Design Automation Conference with EURO VHDL, 1995 San Fransisco ca Moscone center; pp 222-227, 1995.

[80] R. Munden, "ASIC AND FPGA VERIFICATION: A GUIDE TO COMPONENT MODELING", ELSEVIER, 2005.

[81] W. KAFIG, VHDL 101. OXFORD: ELSEVIER, 2011.

[82] B. A. A. Antao, "Architectural exploration for analog system synthesis", IEEE 1995 custom integrated circuits conference, pp 25.4.1-25.4.4, 1995.

[83] IEEE Standard VHDL Synthesis Packages, IEEE-1076.3, 1997 Définit en autre le paquetage Numeric_Std, avec les types Unsigned et Signed.

[84] J. ROSE , A. EL GAMAL and A. SANGIOVANNI-VICENTILLI, "Architecture of Field-Programmable Gate Array" in Proceeding of the IEEE Vol. 81 Issue N° 7, 1993.

[85] IEEE Standard for VHDL Register Transfer Level Synthesis, IEEE 1076.6-1999.

[86] Z. Ait Ouali, "Application des FPGA à la commande d'un moteur asynchrone", thèse de magister de l'université MOULOUD MAMMRI, TIZI-OUZOU, 2010.

[87] A. FAKHFAKH, "Contribution à la modélisation comportementale des circuits radiofréquence" Thèse de doctorat de l'université de BORDEAUX I, 2002.

[88] P. Collet " Un modèle fondé sur les assertions pour le génie logiciel et les bases de données ". Thèse de doctorat de l'université de Nice-Sophia Antipolis, décembre 1997.

[89] J.Tan, "Exploration d'architectures génériques sur FPGA pour des algorithmes d'imagerie multi spectrale", thèse de doctorat de l'université jean Monnet, Juin 2012.

[90] A .Benyamina, "Ordonnancement Hiérarchique Multi-Objectif D'Application Embarquées Intensives (CAO/IAO&Simulation)", thèse de doctorat de l'université d'Oran-Es-Senia, 2009.

[91] S. Jose, ALTERA, Introduction to the Quartus II Software. CA 95134 (408) 544-7000, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/intro_to_quartus2.pdf

[92] M.A. Koulali, "Réseaux de capteurs sans fil: QoS et analyse des performances", édition : Editions universitaires europeennes,152 pages, 2012, ISBN-13: 978-3-8381-8166-0, ISBN-10: 3838181662

[93] K. OUDIDI, "Routage et Qualité de Service dans les réseaux sans fil spontanés", thèse de Doctorat de l'école Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS)-RABAT, juillet 2010.

[94] Command Line Tools User Guide UG628 (v 14.2), July 2012 (This document applies to the following software versions: ISE Design Suite 14.2 through 14.4), http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/devref.pdf.

[95] ISim User Guide, UG660 (v14.1), Avril 2012, https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/plugin_ism.pdf.

[96] R. Benaicha, M. Taibi, "A new technique for accelerating routing information process in communication networks", Revue Synthèse, numéro : 32, Avril 2016, pages 106-114, ISSN : 11114924.

[97] S. Pillement, "Conception d'architectures reconfigurables dynamiquement : Du silicium au système", Thèse de doctorat de l'université de Rennes 1, 2010.

[98] N. Larrieu, "Proposition d'une architecture de gestion du trafic à partir de mesures reposant sur un mécanisme de contrôle de congestion permettant l'optimisation de la QdS dans l'Internet", Thèse de doctorat de l'université de Toulouse, 2005.

[99] M. BENJRAD, "Robustesse par conception de circuits implantés sur FPGA SRAM et validation par injection de fautes", Thèse de doctorat de l'université de Grenoble, 2006.