REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
– MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE –
ECOLE NATIONALE POLYTECHNIQUE.

DEPARTEMENT D'ELECTRONIQUE

MEMOIRE DE MAGISTER

En Electronique

Option : Télécommunications

:THEME

"Contribution à l'implémentation d'un serveur DNS autoritaire en VHDL "

Présenté par
ABID FAROUDJA
Ingénieur d'état en Electronique UMMTO

Le jury est composé de :

M.er R.AKSAS	Professeur .ENP	Président du jury
M.er R.SADOUN	Chargé de cours .ENP	Rapporteur
M.er HADADI	Professeur .ENP	Examinateur
Mme L.HAMAMI	Maître de conférence .ENP	Examinatrice
M.er STIHI	Chargé de cours .ENP	Examinateur

2006

Remerciements

J’adresse mes remerciements à M.er R.SADOUN qui ma confié ce travail et aidé à sa réalisation.

Je remercie également le président et membres du jury qui ont bien voulu juger ce travail.

Je dédie ce travail à mes chers parents,
ma sœur Nadia, toute ma famille et tous mes amis.

ملخص
هذا العمل يهدف أساسا إلى تنفيذ موزع DNS فوق شريحة اعتمادا على RFC 1034 و1035 المرجعين المحددين للبروتوكول DNS.
نقترح خلال هذا العمل حلا لتصميم خدمة DNS على شكل محيط بهدف تطويق هذه الخدمة لجعلها أكثر حماية.قمنا بتشكيل النموذج باستعمال البرنامج التنفيذي VHDL .
النموذج الأول من نوعه الذي تم اقتراحه في هذا العمل بهدف إلى حماية خدمة DNS التي تمثل أحد أعمدة الشبكة ، ووسيلة هامة لتسيير خدمة الانترنيت.

Résumé

Notre travail a consisté en l'implémentation hardware d'un serveur DNS autoritaire, en se basant sur les RFC (*Request For Comments)* 1034, 1035 ayant défini les spécifications du protocole de résolution de nom. Nous proposons dans ce travail une solution architecturale permettant l'implémentation matérielle de ce type de protocole. Une description grâce au langage VHDL nous a permis d'envisager une implémentation hardware dont l'intérêt est de permettre un isolement physique de ce service. L'objectif est sa sécurisation vu qu'il représente une ressource critique pour le fonctionnement de plusieurs applications Internet.

Mots clés : Serveur, Résolveur, protocole DNS, Implémentation hardware, RFC, VHDL.

Abstract

Our work consists of the hardware implementation of authoritative DNS server, while being based on the RFC (Request For Comments) 1034, 1035 defining the specifications of protocol DNS. We propose in this work an architectural solution allowing the hardware implementation of DNS service.
A material description of this service was made with language VHDL for implementation.
Our interest is to allow an implementation in the form of circuit .The solution proposed allows primarily the physical insulation of this service. The objective is the security of the one of the services more used in Internet networks, and who is a critical resource for the correct operation of Internet applications.

Key words : Serveur, Resolveur, protocole DNS, Hardware implémentation, RFC, VHDL.

TABLE DES MATIERES

Chapitre3 Méthodologie de développement d'un serveur DNS en VHDL

Chapitre4 Modélisation et implémentation VHDL d'un serveur DNS autoritaire

LISTE DES FIGURES

Acronymes

ARPA:	Advanced Research Project Agency
ASICs:	Application Specific Integrated Circuits.
BIND :	The Berkeley Internet Name Domain
DNS:	Domain Name System
DNSsec:	Domain Name System Security Extensions
l'IETF:	Internet Engineering Task Force
FPGA:	Field Programmable Gate Array
HDL:	Hardware Description Langage
IP:	Internet Protocol.
ISO:	International Standards Organisation.
OSI:	Open System Interconnection.
RFC:	Request For Comments.
SRI-NIC:	Stanford Research Institute- Network Information Center
SOA	Start Of Authority
RR:	Resource Records
TCP:	Transport Control Protocol.
UDP:	User Datagramme Protocol.
VHDL:	Vhsic Hardware Description Language

Introduction Générale

Sur le réseau Internet, les machines sont identifiées par des adresses IP. Néanmoins, ces adresses ne sont pas faciles à manipuler; c'est pourquoi on utilise une abstraction plus accessible à travers des noms d'hôtes.

Jusqu'en 1984, cette association entre les noms des machines connectées au réseau Internet et leurs adresses était établie à travers un simple fichier appelé table de conversion géré par une autorité centrale, le SRI-NIC . Cette approche a l'inconvénient majeur de nécessiter la mise à jour des tables de tous les ordinateurs dans le cas d'insertion ou de modification d'un nom de machine. Avec l'explosion de la taille des
réseaux et de leurs interconnexions cette manière de procéder était devenue totalement inadaptée.

Pour répondre à cette contrainte, une solution a été proposée par Paul Mockapetris et John Postel; le DNS ou «Domain Name System». Il s'agit d'un système distribué de gestion des noms utilisant une base de données distribuée reflétant un espace de nommage hiérarchique et arborescent de noms et garantissant l'unicité de ces derniers. Le fonctionnement sera assuré par un ensemble d'ordinateurs répartis; chacun devant assurer le service de résolution de noms des hôtes qui sont sous son autorité ou coopérer avec d'autres serveurs dans le cas où les données sont distantes.
Le protocole DNS a été conçu suivant une philosophie toute naturelle selon laquelle les données DNS sont publiques avec un accès totalement libre pour assurer

l'interconnexion logique d'un ou plusieurs ordinateurs basée sur l'abstraction de leurs adresses. Le bon fonctionnement d'Internet est donc totalement tributaire de la fiabilité de ce système. Cette fiabilité dépend aussi bien du matériel que du logiciel implémenté pour assurer ce service. Une défaillance d'un simple lien physique sur un serveur DNS peut isoler complètement un campus ou les clients d'un provider donné si des mécanismes de redondance ne sont pas mis en jeu. Il dépend aussi de la fiabilité de fonctionnement des applications implémentées. Que dire de l'authenticité et de l'intégrité des données qui y sont gérées.

Toutes les implémentations connues jusqu'à ce jour sont sous forme logicielle. Assurer la fiabilité de fonctionnement de ce système sous l'angle de l'implémentation logicielle revient à assurer l'intégrité de fonctionnement des processus qui en découlent. Une simple infection virale deviendra dramatique à la bonne interconnexion des systèmes auquel il participe. Pour remédier à ce type de contrainte et assurer ainsi une meilleure fiabilité, il nous a été proposé une contribution à une implémentation hardware de ce service. L'intérêt est double; garantir l'intégrité du service et donc une bien meilleure continuité du service d'une part et d'autre part obtenir de meilleures performances de fonctionnement locales réduisant ainsi les latences liées au système de traitement. Il est clair aussi que cette forme d'implémentation ne pourra que mieux garantir l'intégrité des données gérées localement vis à vis d'événements exogènes.

Comme il va être décrit plus loin, l'implémentation que nous avons conçue et réalisée est constituée d'un ensemble de blocs fonctionnels communicants décrits à l'aide d'un langage de description; la cible technologique choisie étant un FPGA. L'architecture établie a tenu compte des définitions des règles de normalisation édictées par la communauté Internet.

Notre mémoire a été subdivisé en quatre chapitres.
Dans le premier, nous exposerons le concept DNS et nous détaillerons les mécanismes de résolution ainsi que les principes généraux permettant la compréhension du fonctionnement de ce système. Dans le second chapitre, nous

décrirons l'algorithme du résolveur ainsi que celui du serveur. Nous donnerons les organigrammes qui illustrent les étapes de leurs fonctionnements.

Nous exposerons dans le troisième chapitre la méthodologie que nous proposons pour l'implémentation hardware d'un service DNS, et nous détaillerons les différentes étapes du flot de conception suivi.

Le dernier chapitre traitera de l'implémentation proprement dite. Nous commencerons par décrire la solution architecturale proposée, pour l'implémentation du service DNS en VHDL, en détaillant l'ensemble des blocs fonctionnels qu'elle contient.

Ensuite, nous présenterons les résultats de simulation obtenus pour chaque bloc et les résultats de leurs interconnections.

1 Système de noms de domaines

1.1 Introduction

L'association des noms de système Internet et des adresses IP [3] est une tâche qui demande beaucoup de réflexion .Internet à connue une énorme croissance, et le système initial où les correspondances entre nom d'hôtes et adresses IP étaient sauvegardées dans un seul fichier local s'est vite avéré peu pratique. Un nouveau système était nécessaire afin de gérer les milliers d'ordinateurs du Net. L'objectif a alors été de permettre une résolution qui consiste à assurer la conversion des noms d'hôtes en adresses IP ou inversement. La solution actuellement utilisée pour répondre à ce besoin est le DNS (**D**omain **N**ame **S**ystem) ; ce qui va être introduit dans le présent chapitre.

1.2 Historique

Jusqu'en **1984**, étant donné que les réseaux étaient très peu étendus, c'est-à-dire que le nombre d'ordinateurs connectés à un même réseau était faible; toutes les associations entre les noms de machines connectées au réseau Internet et leur(s) adresse(s) étaient contenues dans un simple fichier appelé *tables de conversion manuelle* (fichiers généralement nommés *hosts* ou *hosts.txt*).

Ces fichiers "HOSTS" contenaient une liste de noms et les adresses IP associées. Ce fichier était géré par une autorité centrale, le SRI-NIC chargée de recueillir les mises à jour et de diffuser régulièrement une version actualisée.

Avec l'explosion de la taille des réseaux et de leurs interconnexions les collisions entre noms ainsi que les difficultés inhérentes à une gestion centralisée (mises à jour et diffusion d'informations) ont rendu ce modèle inadapté. D'où l'apparition d'un système non centralisé de gestion des noms d'hôtes en relation avec leur(s) adresse(s) IP. Ce système a été dénommé DNS. L'implémentation qui a prédominé jusqu'à nos jours a consisté en un espace hiérarchique de noms permettant de garantir l'unicité d'un nom d'hôte dans une structure arborescente.

L'idée donc qui a motivé la création de DNS était de construire un espace de noms cohérent et quasi infini grâce à un modèle distribué, arborescent, robuste et performant.

Les toutes premières règles de nommage ont été définies dans le standard RFC 822 pour la messagerie ARPANET (mars **1982**).

Au cours des années **1983-1984**, Paul Mockapetris et John Postel proposèrent et développèrent une solution utilisant une base de données distribuée :Le système de noms de domaines . Les RFCs 882 et 883 sont alors devenues obsolètes. Le protocole associé a été défini et standardisé par l'IETF à travers les RFC 1034 et 1035 (**1987**).

Avec la généralisation du réseau et la multiplication des usages, le DNS est devenu l'un des piliers du fonctionnement de l'Internet d'où le besoin de sa sécurisation. Les extensions de sécurité au protocole DNS (le DNSsec) ont alors été spécifiées par l'IETF à travers la RFC 2535 (**1999**)**.**

1.3 Système de Noms de Domaines (DNS)

C'est un système de noms pour les ordinateurs organisé selon une hiérarchie de domaines. Il s'agit d'une base de données (ou arbre de nommage) répartie contenant des informations utilisées pour traduire des noms de domaines en une adresse chiffrée dite adresse IP couplée à un protocole de communication. Ce système est utilisé dans les réseaux TCP/IP [3] pour localiser des ordinateurs et des services à l'aide de noms conviviaux. Lorsqu'un utilisateur entre un nom DNS dans une application, le service DNS peut résoudre ce nom en une autre information qui lui est associée; une adresse IP. Il s'agit de la fonction la plus connue du protocole DNS : l'assignation des noms à des adresses IP.

Les usagers d'Internet conservent et mettent à jour la partie de la base de données qui les concerne pour un usage local ou coopératif.

1.3.1 Les trois composantes du DNS

Le DNS a été organisé en trois composantes [1].

a- L'espace des noms et la liste des ressources

L'espace des noms et la liste des ressources définissent un ensemble de noms et les données qui leur sont associées, le tout étant organisé sous la forme d'un arbre. Chaque nœud et chaque feuille de cet arbre contiennent donc un certain nombre d'informations. Ces informations sont stockées dans des structures appelées « Resource Records ». Les RRs ou les « Resource Records » sont les briques de base de l'information DNS. C'est une association entre un nom de domaine et une information se rapportant à ce nom (adresse IP, relais de messagerie, ...). Ces informations seront consultées grâce à des requêtes. Ainsi, pour accéder à une information donnée, on précisera dans la requête le nom de domaine concerné et le type d'information désirée.

b- Les serveurs de noms

Les serveurs de noms implémentent des applications qui assurent la traduction des noms en adresses IP et réciproquement. Pour cela, ils possèdent des informations sur une partie de la structure de l'arbre de nommage et sur les données qui lui sont associées. Un serveur de nom peut mémoriser des informations au sujet de n'importe

quelle partie de l'arbre, mais en général, il contient plutôt des informations complètes sur une partie de l'arbre. Il peut posséder également des références vers d'autres serveurs susceptibles de fournir des informations sur le reste de l'arbre. Lorsqu'un serveur a la connaissance complète sur une partie de l'arbre, on dit qu'il a autorité sur cette partie.

c- Les résolveurs de noms

Les résolveurs de noms sont des applications qui, à la suite d'une demande de la part d'un client, obtiennent l'information recherchée auprès des serveurs de noms. Un résolveur doit donc pouvoir interroger un serveur de nom et utiliser l'information reçue pour répondre à la demande ou pour interroger un autre serveur susceptible de connaître la réponse. Le plus souvent, un résolveur se présentera sous la forme d'une routine système directement accessible par les programmes utilisateurs.

1.4 Structure de l'espace de nommage

L'architecture du DNS repose sur une structure arborescente (figure 1.1)

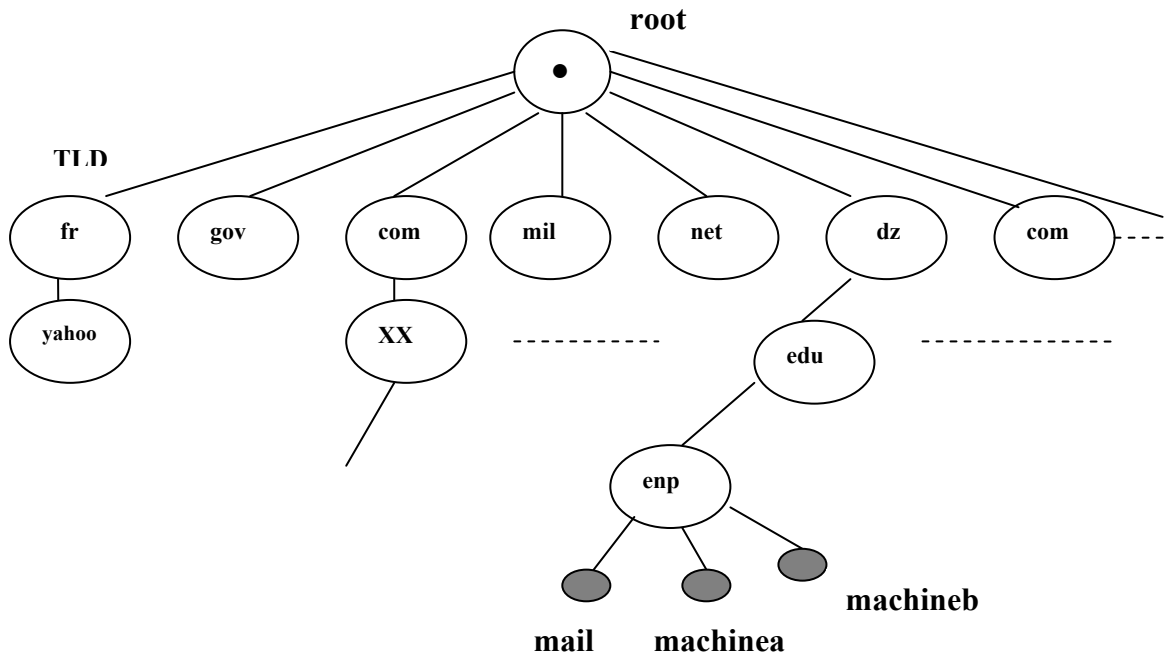


Figure 1.1: Structure de l'espace des noms

L'arbre DNS est composé de plusieurs domaines : un sous arbre partant d'un noeud jusqu'aux feuilles. Chaque domaine est composé de zones.

Une zone est la partie descriptive des informations DNS d'un sous arbre. Ces informations sont contenues dans un fichier de zone stocké sur des serveurs dits autoritaires. Ces derniers sont en charge de la mise à jour et de la diffusion des informations qu'ils gèrent.

Un nom de domaine est structuré, il fournit souvent des informations sur le type d'entité qu'il représente. Il est représenté par la succession d'étiquettes (*labels*) séparées par des points et rencontrées sur le chemin reliant le noeud à la racine.

Les noms de domaines situés au même niveau hiérarchique doivent être uniques. Par exemple, il ne peut y avoir qu'un seul `com` au niveau supérieur de la hiérarchie et un seul `xx.com` au niveau suivant de la hiérarchie.

Les principaux domaines de premier niveau ou **TLD** (*Top Level Domain)* sont subdivisés en deux catégories :

- Les domaines dits « génériques », appelés **gTLD** (*generic TLD*) : sont des noms de domaines génériques de niveau supérieur proposant une classification selon le secteur d'activité.
- Les domaines dits «nationaux », appelés **ccTLD** (country code TLD), ils correspondent aux différents pays et leurs noms correspondent aux abréviations des noms de pays définies par la norme ISO 3166.

La base de données DNS a été conçue suivant trois caractéristiques principales [6]:

– Elle est hiérarchique; ce qui lui confère l'appellation d'arbre DNS où chaque nom de domaine est représenté par un nœud de l'arbre; la racine étant représentée par "**.**". Cette structuration en arbre inversé permet de relier la racine à un nœud quelconque de l'arbre par un chemin unique.

Un domaine est tout simplement un sous arbre de l'arbre DNS débutant à un noeud spécifique et recouvrant l'arborescence située en dessous de ce point. Le domaine *enp.edu.dz* par exemple est inclus dans le domaine *dz.*

– Elle est distribuée; elle est répartie sur un grand nombre de serveurs, chacun de ces serveurs étant en charge d'un sous arbre de l'arbre DNS et des informations correspondantes. On évite ainsi la lourdeur d'un système centralisé où toutes les requêtes sont traitées par une base de données unique, même si celle-ci est répliquée sur plusieurs autres serveurs.

Cette décentralisation permet d'augmenter la flexibilité du système. Une administration locale est plus à elle même de gérer les mises à jour des informations du sous arbre dont elle est en charge. C'est pour cela qu'au sein d'un domaine, on choisit souvent de transférer la responsabilité de certains sous ensembles de noms; c'est ce qu'on appelle une délégation. Cela a pour conséquence la création d'une nouvelle zone dite zone fille de la zone parente.

– **Elle est redondante** ; la décentralisation des responsabilités s'accompagne évidemment d'une redondance des données pour augmenter la robustesse. Chaque zone est en fait prise en charge par plusieurs serveurs répartis géographiquement et topologiquement.Parmi les serveurs autoritaires pour une zone, l'un d'entre eux, dit serveur primaire, est chargé de transmettre une réplique du fichier de zone chaque fois qu'il est modifié aux autres serveurs dits secondaires. Une telle opération est appelée transfert de zone.

1.5 Le message DNS

1.5.1 Format

Les requêtes et les réponses DNS sont transportées dans un message de format standardisé [2]. Ce format définit un entête contenant un certain nombre de champs fixes toujours présents et quatre sections pour transporter la question et les RRs comme le montre la figure ci après.

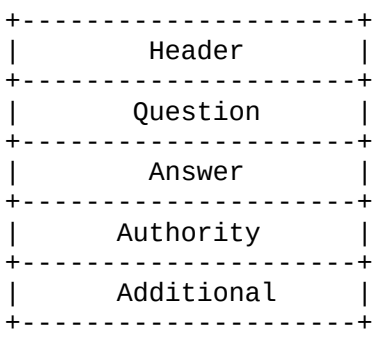


Figure 1.2: Format du message DNS

1.5.1.1 L'entête

La figure suivante illustre la structure de l'entête basée sur 12 octets, les significations des champs véhiculés par l'entête sont données dans le tableau 1.

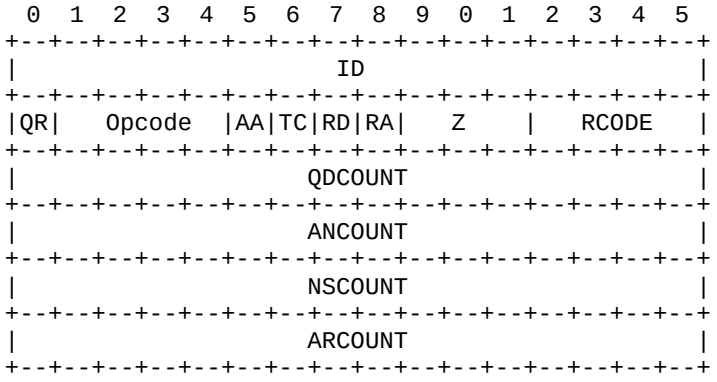


Figure 1.3: Format de l'entête du message DNS

Champs	Taille	significations
ID	16 bits	doit être recopié lors de la réponse permettant à l'application de départ de pouvoir identifier le datagramme de retour.
Qr	1 bit	permet d'indiquer s'il s'agit d'une requête (0) ou d'une réponse (1).
Opcode	4 bits	permet de spécifier le type de requête : Opcode =0 - Requête standard (Squery) Opcode =1 - Requête inverse (Iquery) Opcode =2 - Status d'une requête serveur (Status) Opcode =3-15 - Réservé pour des utilisations futures
Aa	1 bit	signifie "Authoritative Answer". Il indique une réponse d'une entité autoritaire.
Tc	1 bit	indique que ce message a été tronqué
Rd	1 bit	permet de demander la récursivité en le mettant à 1.
Ra	1 bit	indique que la récursivité est autorisée.
Z	1 bit	est réservé pour une utilisation future. Il doit être placé à 0 dans tout les cas.
Rcode	4 bits	indique le type de réponse. Rcode =0 - Pas d'erreur Rcode =1 - Erreur de format dans la requête Rcode =2 - Problème sur serveur Rcode =3 - Le nom n'existe pas Rcode =4 - Non implémenté Rcode =5 – Refus Rcode =6-15 – Réservés
Qdcount	16 bits	spécifie le nombre d'entrée dans la section "Question".
Ancount	16 bits	spécifie le nombre d'entrée dans la section "Réponse".
Nscount	16 bits	spécifie le nombre d'entrée dans la section "Autorité".
Arcount	16 bits	spécifie le nombre d'entrée dans la section "Additionnel".

Tableau 1 : Signification des champs de l'entête

1.5.1.2Format de la section question

Le format de la section question est comme suit :

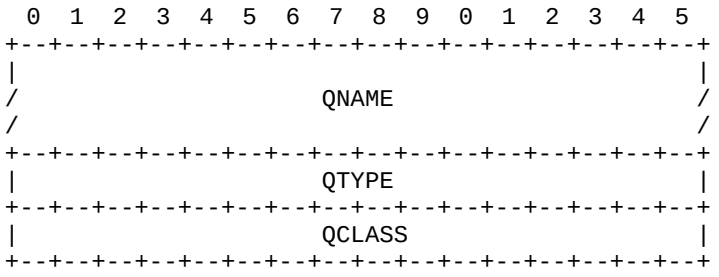


Figure 1.4: Format de la section question

La section **Question** contient la question transmise par un client. Elle est constituée de trois champs:

-Qname correspondant au nom d'hôte ou nom de domaine sur lequel on cherche des renseignements indiqué sous la forme d'un nom de domaine complet (FQDN, *Fully Qualified Domain Name*).

-Qtype représentant le type de renseignements recherchés.

-Qclass représentant la classe de la requête.

Ces trois informations définissent une question à laquelle le serveur doit répondre.

Les trois autres sections contiennent les RRs :

*** Answer,** contient les RRs qui répondent à la question.

***Authority,** contient des RRs qui indiquent des serveurs ayant une connaissance complète de cette partie du réseau.

***Additional,** contient des RRs supplémentaires pouvant être utiles pour exploiter les informations contenues dans les autres sections.

1.5.2 Les RR

La base de données des serveurs de noms (fichier de domaine et fichiers de résolution inverse) est constituée "d'enregistrements de ressources"ou "Resource Records" (RRs). Ces enregistrements sont répartis en classes. La seule classe d'enregistrement usuellement employée est la classe Internet (IN).

Voici les différents champs d'un RR.

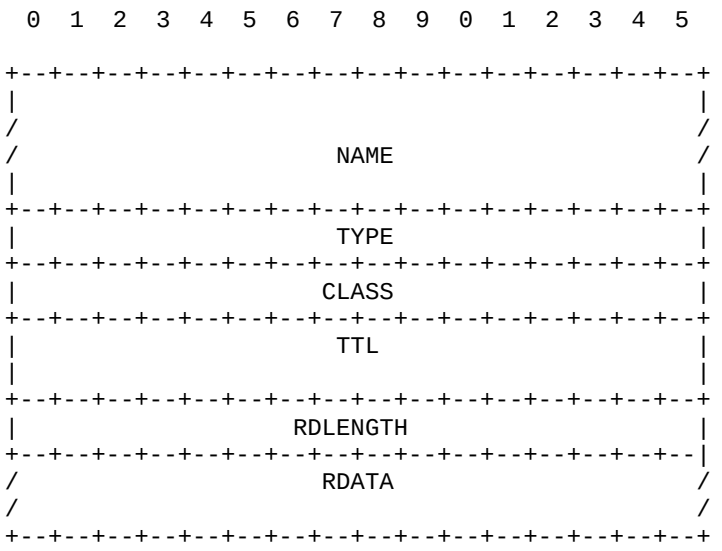


Figure1.5: Format d'un RR

Name	Nom du domaine où se trouve le RR
Type	sur 16 bits, spécifie quel type de donnée est utilisé dans le RR.
Classe	16 bits, identifie une famille de protocoles
TTL	C'est la durée de vie des RRs (32 bits, en secondes), utilisée par les résolveurs de noms lorsqu'ils ont un cache pour connaître leurs durées de validité.
Longueur	Sur 16 bits, ce champ indique la longueur des données suivantes
Données	identifie la ressource, ce que l'on met dans ce champ dépend évidemment du type de ressources que l'on décrit

Tableau 2 : Significations des champs constituant un RR

Un RR est composé de six champs de longueur variable, leurs significations sont données dans le tableau ci-dessus.

1.5.3 Le transport

1.5.3.1 Utilisation d'UDP

La plupart des échanges DNS sont effectués en UDP [8] sur le port 53. Les datagrammes DNS en UDP sont limités à 512 octets (valeur représentant les données sans les entêtes UDP et IP). Les datagrammes plus longs doivent être tronqués ; cette action est précisée par la suite à l'aide du champ Tc.
L'avantage d'UDP dans ce type de requêtes est sa rapidité. Même s'il ne garantit pas l'arrivée des informations, il se révèle être très rapide. L'utilisation d' UDP n'est pas recommandée pour le transfert de zone, mais uniquement pour les requêtes standards.

1.5.3.2 Utilisation de TCP

Le port serveur utilisé pour l'envoi des datagrammes en TCP [8] est 53. Il gère uniquement les requêtes longues, les transferts de zones, etc. Le datagrammes inclus alors un champ de deux octets nommé "longueur". Il permet de spécifier la longueur total des données indépendamment de la fragmentation.

1.6 Fonctionnement de DNS

1.6.1 Requête

La principale activité d'un serveur de noms est de répondre aux requêtes standards. La requête et sa réponse sont toutes deux véhiculées par un message standardisé décrit dans la RFC 1035. Les requêtes sont des messages envoyés aux serveurs de noms en vue de consulter les données stockées sur ce serveur.
Chaque message de requête envoyé par le client contient trois informations (QTYPE, QCLASS, et QNAME) décrivant respectivement le(s) type(s) et la classe de l'information souhaitée et quel nom de domaine cette information implique.

1.6.1.1 Structure des requêtes

Parmi les champs fixes on trouve un champ très important de 4 bits désignés par *opcode*. Il permet de donner des informations sur le type de requête.
Une requête comporte aussi les quatre sections décrites précédemment.
Notons que les sections answer, authority et additional sont vides, pour une requête.
Voici un exemple de requête où l'on souhaite connaître le nom du serveur de courrier gérant le domaine ENP.EDU.DZ

Header	OPCODE=SQUERY
Question	QNAME=ENP.EDU.DZ, QCLASS=IN, QTYPE=MX
Answer	Vide
Authority	Vide
Additionnal	Vide

La réponse renvoyée par le serveur peut prendre la forme :

Header	OPCODE=SQUERY, RESPONSE, AA			
Question	QNAME=ENP.EDU.DZ, QCLASS=IN, QTYPE=MX			
Answer	ENP.EDU.DZ IN	MX	10	MAIL.ENP.EDU.DZ.
Authotity	Vide			
Additionnal	MAIL IN	A	193.194.70.90	

1.6.2 Mécanisme de résolution

D'une manière générale, le traitement d'une requête DNS est un processus pouvant impliquer au plus deux phases :

- Une requête de nom formulée au niveau d'un ordinateur client est transmise à un résolveur en vue d'être résolue.
- Lorsque la requête ne peut pas être résolue localement (au niveau du résolveur), des serveurs DNS nécessaires pour résoudre le nom seront interrogés.

Ces deux phases sont expliquées plus en détail dans ce qui suit.

1ère partie : utilisation du résolveur local

Le graphique qui suit présente le processus complet de traitement d'une requête DNS.

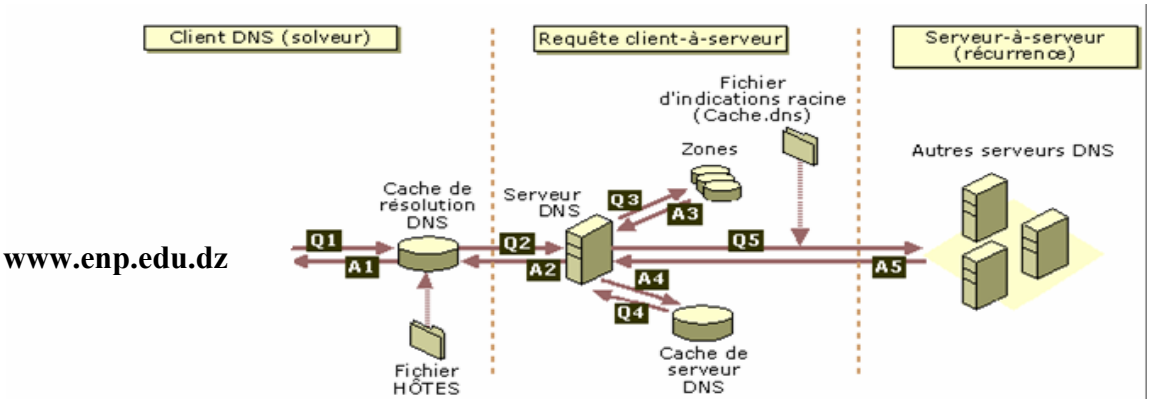


Figure1.6: Mécanismes de résolution

Comme l'indiquent les premières étapes de ce processus de résolution, un nom de domaine DNS est utilisé dans un programme de l'ordinateur local. La requête est ensuite transmise au service DNS local afin d'être résolue à l'aide des informations mise en cache localement. Si la requête de nom peut être résolue, une réponse est constituée et le processus se termine.

Le cache de résolution local peut inclure des informations de résolution issues de deux sources :

- Si un fichier Hosts est configuré localement, tout mappage nom d'hôte-adresse contenu dans ce fichier est préchargé dans le cache au démarrage du service client DNS.
- Les enregistrements de ressources obtenus en réponse à de précédentes requêtes DNS sont ajoutés au cache et conservés pendant un temps préétabli.

Si la requête ne correspondant à aucune entrée du cache, le processus de résolution se poursuit et le client interroge un serveur DNS auquel il est rattaché pour résoudre une requête donnée.

2ème partie : interrogation d'un serveur DNS

Comme l'indique le graphique précédent, le client interroge un serveur DNS auquel il est rattaché.

A réception de la requête, le serveur DNS commence par vérifier s'il peut y répondre en utilisant des informations d'enregistrements de ressources présentes dans une zone configurée localement. Si la requête de nom correspond à un enregistrement de ressource dans les informations de la zone locale, le serveur répond.

Dans le cas contraire, le serveur vérifie s'il peut résoudre ce nom en utilisant les informations mises localement en cache à partir de requêtes antérieures. Si une correspondance est trouvée, le serveur envoie ces informations et la requête prend fin.

1.6.2.1 Modes de résolution

1.6.2.1.1 Mode récursif

Si aucune réponse à la requête de nom n'est trouvée sur le serveur par défaut - dans son cache ou ses informations de zone- le processus de requête se poursuit, et le premier serveur DNS prend le rôle de résolveur et interroge ou contacte d'autres serveurs DNS au nom du client effectuant la requête, de manière à complètement résoudre la requête pour renvoyer une réponse au client par la suite. On dit dans ce cas que le serveur est configuré en mode récursif.

La récursivité est utilisée pour résoudre complètement un nom. Ce processus implique l'assistance d'autres serveurs DNS. Par défaut, le service client DNS demande au serveur d'utiliser un processus de récursivité pour résoudre complètement les noms pour son compte avant de renvoyer une réponse. Le processus de récursivité est illustré sur le graphique qui suit.

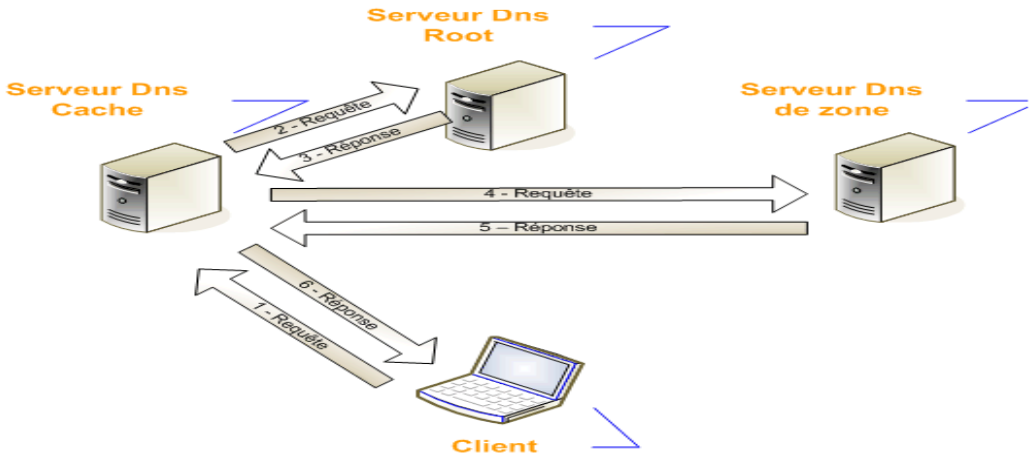


Figure1.7: Mode récursif

Pour pouvoir correctement utiliser le processus de récursivité, le serveur DNS a besoin de connaître les coordonnées des autres serveurs DNS de l'espace de noms de domaines DNS. Ces informations sont fournies sous la forme « d'*indications racines* », une liste

d'enregistrements de ressources préliminaires qui peut être utilisée par le service DNS pour localiser d'autres serveurs DNS qui font autorité pour la racine de l'arborescence de l'espace de noms de domaines. Les serveurs « racines » font autorité sur la racine et les domaines de premier niveau dans l'arborescence.

L'utilisation du mode récursif est limitée aux cas qui résultent d'un accord négocié entre le client et le serveur. Cet accord est négocié par l'utilisation de deux bits particuliers des messages de requête et de réponse :

Le bit *Ra* (Récursion admissible) est marqué ou non par le serveur dans toutes les réponses. Ce bit est marqué si le serveur accepte à priori de fournir le service récursif au client, que ce dernier l'ait demandé ou non. Autrement dit, le bit *Ra* signale la disponibilité du service plutôt que son utilisation.

Les requêtes disposent d'un bit *R*d (pour "récursion désirée"). Ce bit indique que le requérant désire utiliser le service récursif pour cette requête. Les clients peuvent demander le service récursif à n'importe quel serveur de noms, bien que ce service ne puisse leur être fourni que par les serveurs qui auront déjà marqué leur bit *Ra*.

Le mode récursif est mis en œuvre lorsqu'une requête arrive avec un bit *Rd* marqué sur un serveur annonçant disposer de ce service, le client peut vérifier si le mode récursif a été utilisé en constatant que les deux bits *Ra* et *Rd* ont été marqués dans la réponse.

Le mode récursif, présente des avantages au niveau des performances du serveur DNS. Par exemple, au cours du processus de récursivité, le serveur DNS effectuant la recherche récursive obtient des informations sur l'espace de noms de domaines DNS. Ces informations sont mises en cache par le serveur et peuvent être réutilisées pour accélérer la réponse donnée à des requêtes ultérieures.

1.6.2.1.2 Mode itératif

Une autre manière de procéder donne la possibilité au client de contacter d'autres serveurs DNS pour résoudre une requête donnée. Dans ce cas, il utilise des requêtes supplémentaires distinctes reposant sur des réponses de référence d'un autre serveur qui sera plus susceptible de disposer de l'information demandée.

La figure suivante illustre ce mode de résolution.

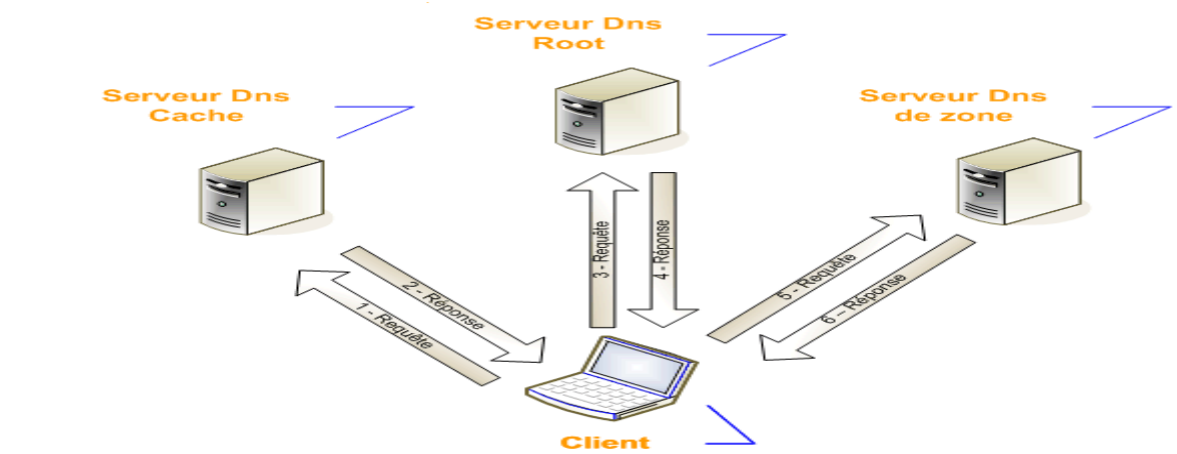


Figure1.8: Mode itératif

L'itération est le type de résolution de nom utilisé entre les clients et les serveurs DNS dans les conditions suivantes :

- Le client demande l'utilisation de la récursivité, mais celle-ci est désactivée sur le serveur DNS.
- Le client ne demande pas l'utilisation de la récursivité lors de l'interrogation du serveur DNS.

Une requête itérative formulée par un client indique au serveur DNS que le client attend que ce dernier lui fournisse immédiatement la meilleure réponse possible, sans contacter d'autres serveurs DNS.

Dans une requête itérative, un serveur DNS répond à un client en utilisant ses propres connaissances sur l'espace de noms concernant les données de nom spécifiées dans la requête. Par exemple, si un serveur DNS d'un intranet reçoit une requête d'un client local portant sur « www.xx.yy.zz», il peut renvoyer une réponse à partir de son cache de noms. Si le nom faisant l'objet de la requête n'est pas stocké dans le cache de noms du serveur, ce dernier peut répondre par une référence, c'est-à-dire une liste d'enregistrements de ressources de type NS et A pour d'autres serveurs DNS plus proches du nom spécifié dans la requête du client.

Lorsqu'une réponse de référence est renvoyée, le client DNS prend la responsabilité de poursuivre les requêtes itératives vers d'autres serveurs DNS configurés pour la résolution souhaitée. Dans la plupart des requêtes itératives, un client utilise la liste des serveurs DNS configurée localement pour contacter d'autres serveurs de noms dans l'espace de noms DNS si son serveur DNS principal ne peut pas résoudre la requête. Le

mode non récursif est approprié si le résolveur est capable de façon autonome de poursuivre sa recherche et est capable d'exploiter l'information supplémentaire qui lui est envoyée pour l'aider à résoudre sa requête.

1.6.2.2 Fonctionnement de la mise en cache

Lorsque les serveurs DNS traitent les requêtes clientes à l'aide de la récursivité ou de l'itération, ils acquièrent de nombreuses informations sur l'espace de noms DNS. Ces informations sont ensuite mises en cache sur le serveur.

Par la suite, lorsque d'autres clients formulent de nouvelles requêtes portant sur des informations d'enregistrement de ressources correspondant à des enregistrements de ressources mis en cache, le serveur DNS peut utiliser ces informations pour y répondre.

La mise en cache permet d'accélérer la résolution DNS pour les requêtes ultérieures portant sur des noms très utilisés, tout en réduisant de manière significative le trafic des requêtes DNS sur le réseau.

1.7 Failles du DNS

Le protocole DNS a été conçu suivant une philosophie selon laquelle les données DNS sont publiques et l'accès à ces données est universel. Les problèmes de sécurité ne touchent pas au domaine de la confidentialité des données.

En revanche, un certain nombre de failles de sécurité peuvent perturber le bon fonctionnement du DNS et ainsi aller à l'encontre du cahier des charges du service DNS (qui doit, autant que possible, garantir la disponibilité, l'authenticité et l'intégrité des données DNS).

Les buts des attaques sur le DNS sont multiples et les conséquences peuvent aller de l'indisponibilité d'une ressource au vol d'informations critiques (emails, mots de passes etc..).

Les messages ont une structure simple et sont transportés dans un simple paquet UDP, non chiffré et non signé, ce qui rend très facile la génération de faux paquets et leur insertion dans le trafic DNS.

On peut classer les objectifs des attaquants comme suit :

–perturber, ralentir ou bloquer le service DNS sur une partie de l'arbre.

– rediriger les utilisateurs ou leurs communications (emails) à leur insu vers des hôtes contrôlés par le pirate ;

– récupérer des informations critiques (logins/mots de passe) en se faisant passer pour le serveur (telnet, ftp, etc..) auquel l'utilisateur croit se connecter.

Toutes ces attaques ont généralement une première étape en commun que l'on désigne souvent par le terme de DNS Spoofing (usurpation d'identité) ou DNS Hijacking.

1.7.1 DNS spoofing

Le DNS Spoofing est un terme générique pour désigner une attaque dont l'objectif est de rediriger, à leur insu, des internautes vers des sites pirates. Le but de l'attaquant est donc de répondre à la requête d'un utilisateur avant le serveur qui est supposé répondre. Il pourra ainsi fournir à la victime les informations de son choix.

Il existe deux principales attaques de type DNS Spoofing : le DNS ID Spoofing et le DNS Cache Poisoning. Concrètement, le but de l'attaquant est de faire correspondre l'adresse IP d'une machine qu'il contrôle à un nom réel et valide d'une machine publique.

1.7.1.1 Attaques communes aux clients/serveurs

DNS ID « Spoofing »

Les serveurs DNS se comportent comme des clients quand ils ne possèdent pas l'information en cache. Cette particularité nous laisse percevoir que les faiblesses du protocole, cote client, peuvent directement se répercuter côte serveur. Toute la sécurité est donc basée sur un "champs " de 2 octets (en fait un champs de l'entête du protocole DNS) : le DNS ID.

Cette attaque peut être menée de deux façons distinctes, selon les informations dont nous disposons :

a)-DNS ID connu

L'attaque consiste à récupérer ce numéro d'identification (en sniffant, quand l'attaque est effectuée sur le même réseau physique, ou en utilisant une faille des systèmes d'exploitation ou des serveurs DNS qui rendent prédictibles ces numéros) pour pouvoir envoyer une réponse falsifiée avant le serveur DNS. Le seul challenge est de répondre avant le serveur DNS légitime car seule la première réponse est considéré par le client.

b)-DNS ID inconnu [6]

Deviner les requêtes et IDs de la victime : souvent l'attaquant n'est pas présent sur le même réseau physique que la victime, il n'a donc pas directement accès aux requêtes et ID associées de la victime. Dans ce cas, le but de l'attaquant va être de forcer la victime à poser une question donnée et deviner l'ID utilisé.

1.7.1.2 Cache « poisoning »

C'est en effet une pollution de cache : les serveurs récursifs, qui jouent un rôle central en ce qui concerne les performances du modèle DNS, sont de fait des victimes idéales pour les attaquants puisqu'ils gardent en cache les données pendant la durée de leur TTL. Si un attaquant arrive à polluer un cache, il multiplie donc le nombre d'utilisateurs potentiellement victimes d'informations corrompues.
Le "DNS cache poisoning" consiste à corrompre le cache d'un serveur DNS, de ce fait, toutes les requêtes ultérieures qui y seront faites, pointeront vers une machine illégitime.

L'interprétation du champ "additional record" permet à un serveur interrogé, de répondre avec plus d'enregistrements que demandé.
Ce champ a été implémente afin de compléter une requête sur un même domaine. En revanche, ces informations ne devraient pas concerner de domaine extérieur. C'est cette dernière considération qui n'a pas été prise en compte sur de nombreux serveurs (les serveurs DNS "Windows 2000 server" par exemple, d'anciennes versions de Bind...) Cette attaque est triviale. Il suffit simplement d'avoir le control sur un serveur DNS public, pour faire tourner un faux serveur DNS qui permettra de rajouter un champ"additional record".

L'attaque se déroule en plusieurs étapes :

- L'attaquant envoie une requête vers le serveur DNS cible demandant la résolution du nom d'une machine du domaine : domaine.com (ex.: www.domaineX.com).
- Le serveur DNS cible relaie cette requête à ns.domaineX.com (puisque c'est lui qui a autorité sur le domaine domaineX.com)
- Le serveur DNS de l'attaquant (modifié pour l'occasion) enverra alors, en plus de la réponse, des enregistrements additionnels (dans lesquels se trouvent les informations falsifiées à savoir un nom de machine publique associé à une adresse IP de l'attaquant).
- Les enregistrements additionnels sont alors mis dans le cache du serveur DNS cible.
- Une machine faisant une requête sur le serveur DNS cible demandant la résolution d'un des noms corrompus aura pour réponse une adresse IP autre que l'adresse IP réelle associée à cette machine.

1.7.2 Le principe de DNSsec

Il est donc devenu évident qu'un acteur aussi critique et vulnérable que le DNS devait être sécurisé. Les extensions de sécurité au protocole DNS ont été spécifiées à l'IETF en 1999 par le biais de la RFC 2535. Cette première version a jeté les bases d'une méthode de sécurisation du DNS, désormais plus connue sous le nom de DNSsec. DNSsec est un ensemble d'extensions qui permet de sécuriser le protocole DNS au niveau de l'authentification, comme de l'intégrité des données.

1.8 Etat de l'art des implémentations logicielles

Il existe plusieurs implémentations logicielles pour un serveur DNS, On cite :

MaraDNS

MaraDNS est un serveur DNS qui possède toutes les fonctionnalités requises pour être utilisé comme serveur autoritaire et/ou récursif.

Plate-forme : MacOS X, Linux

Langage de programmation : C

PowerDNS

PowerDNS est un serveur autoritaire seulement et doit donc être utilisé conjointement avec un serveur DNS récursif. Sa particularité réside dans le fait que l'information du domaine peut être stockée dans des fichiers de configuration ou dans une base de données.

Plate-forme: Linux, Mac OS X and Windows 2000/XP

Langage de programmation : C++

BIND [5]

BIND est l'implémentation du protocole DNS la plus répandue. On la trouve sur la plupart des systèmes UNIX et MS-Windows. Il fournit une application de référence publiquement redistribuable des composants majeurs du DNS comprenant : un serveur DNS, une bibliothèque de résolution de DNS et des outils pour vérifier les opérations conformes au serveur DNS.

Plate-forme Unix, Linux, Windows

Langage de programmation : C

1.9 Conclusion

Dans ce chapitre, une introduction au système de noms de domaines DNS a été présentée. Elle comprend le concept DNS, la définition des requêtes standards et des réponses qui y sont faites ainsi que le format d'un message DNS et la plupart des formats de classes de données Internet (ex., adresses d'hôtes), tels qu'ils sont définit dans la RFC 1035.

Nous avons abordé aussi le fonctionnement d'un serveur de nom de domaine, et plus précisément des requêtes DNS.

A travers ce qui vient être d'écrit, on constate l'intérêt d'un serveur de nom qui représente un des piliers de fonctionnement d'Internet.

2 Algorithmique de la résolution DNS

2.1 Introduction

On a exposé précédemment les concepts de base du DNS et nous avons passé sous silence de nombreux détails concernant le fonctionnement du résolveur et les algorithmes de résolution associés.
Si la fonction de résolution est analysée de près tenant compte des modes de fonctionnement des serveurs DNS décrits dans le chapitre précédent, il y apparaît qu'on peut distinguer deux fonctions distinctes mais complémentaires : la fonction serveur et la fonction résolveur.

La seconde se distingue de la première en raison de son implémentation aussi bien sur le client que sur le serveur. La fonction serveur quant à elle implémente les modes de résolution tenant compte de l'itérativité ou de la récursivité ainsi que de la gestion des bases de données locales quand le serveur fonctionne en mode autoritaire (responsable sur une ou plusieurs zones). Dans ce chapitre, nous traiterons donc des algorithmes aussi bien du résolveur que du serveur. Nous proposerons des organigrammes illustrant les étapes de leurs fonctionnements nécessaires à l'étape préalable à leur implémentation.

Il est clair que les organigrammes déduits doivent être conformes aux spécifications édictées par les RFC 1034 et 1035 pour assurer la compatibilité avec les serveurs déjà fonctionnels sur la toile. Ces algorithmes ont été décrits par P.Mockapetris [1].

L'ensemble de ce présent chapitre tient compte de cette dernière référence et représente un résumé traduit et complété par des organigrammes de fonctionnement.

2.2 Les Fonctions de résolution

Dans la fonction de résolution de nom se cache en réalité une sous classification. Cette dernière répond au besoin d'interrogation d'un serveur DNS donné. On y trouve les résolutions directe, inverse et générale. Le type de résolution est spécifié par le contenu du message DNS transmis.

2.2.1 Résolution directe

Il s'agit de la translation depuis les noms d'hôtes vers leurs adresses associées. Souvent cette fonction émule les anciens mécanismes utilisant les fichiers HOSTS.TXT.

A partir d'une certaine chaîne de caractères donnée, le client désire obtenir une adresse IP. Dans le contexte du serveur DNS, cette demande se traduit par une requête sur des RR de type A.

2.2.2 Résolution inverse

Il s'agit de la translation depuis une adresse vers un nom d'hôte. A partir d'une adresse IP, le requérant désire une chaîne de caractères donnant le nom d'un hôte donné. Le sens des octets de l'adresse IP sera alors inversé. Les quatre octets ainsi transformés étant alors utilisés comme composante de nom. On les suffixera par la chaîne "IN-ADDR.ARPA". Une requête de type PTR est alors envoyée pour obtenir le RR donnant le nom primaire de l'hôte en question.

2.2.3 Fonction de recherche générale

Cette fonction récupère des informations arbitraires à partir d'un serveur DNS et n'a pas de fonctionnalité antérieure correspondante. Le requérant fournit un QNAME, un QTYPE, et une QCLASS. Il désire obtenir tous les RRs y afférents.

Lorsque le résolveur exécute cette fonction de recherche, on pourra s'attendre à l'une des réponses suivantes :

1– Un ou plusieurs RR comprenant les données demandées.

2– Une erreur de nom (NE). Ceci arrive lorsque le nom présenté n'existe pas. Par exemple, un utilisateur peut très bien avoir mal orthographié un nom d'hôte.

3– Une erreur "donnée non trouvée". Ceci intervient lorsque le nom demandé existe, mais pas les données de type spécifié pour le nom demandé. Par exemple, une requête pour une adresse d'hôte appliquée à un nom de boîte aux lettres retournerait cette erreur, car le nom existe bel et bien, mais on ne trouve pas de RR donnant une adresse d'hôte.

2.3 Le processus de résolution

Comme nous l'avons introduit, le processus de résolution, indépendamment du type de cette dernière, peut être implémenté aussi bien sur un client que sur un serveur DNS. Ce qui est commun aux deux est le résolveur dans le cas où le serveur assure la récursivité. Dans le cas contraire, le serveur est autoritaire sur des zones qu'il doit gérer. On en déduit donc qu'on doit distinguer deux types d'algorithmes tenant compte des assertions précédentes ; un associé au résolveur et l'autre à un serveur de nom (autoritaire ou non).

2.3.1 Le résolveur

Les résolveurs sont des applications qui interfacent les applications utilisateur aux serveurs de noms de domaines. Dans le cas le plus simple, un résolveur reçoit une requête provenant d'une application et renvoie une information sous une forme compatible avec la représentation locale de données du système.

Le résolveur est situé sur la même machine que l'application recourant à ses services, mais devra par contre consulter des serveurs de noms de domaines sur d'autres hôtes.

Comme un résolveur peut avoir besoin de contacter plusieurs serveurs de noms, ou obtenir les informations directement à partir de son cache local, son temps de réponse peut varier selon de grandes proportions, depuis quelques millisecondes à plusieurs secondes.

L'une des raisons les plus importantes qui justifient l'existence des résolveurs est d'éliminer le temps d'acheminement de l'information depuis le réseau, et de décharger simultanément les serveurs de noms, en répondant à partir des données mises en cache en local.

2.3.2 Structure de données utilisées dans un résolveur

En plus de ses ressources propres, le résolveur peut aussi disposer d'accès partagés à des données de zones maintenues par un serveur de noms local. Ceci donne au résolveur l'avantage d'un accès plus rapide aux données, mais impose que le résolveur surveille qu'aucune donnée mise en cache ne vienne masquer une donnée pour cette zone. Dans cette partie, nous appellerons "information locale" la réunion des informations contenues dans le cache et celles de la zone partagée, en sous-entendant que les données "autorisées" seront toujours utilisées de préférence à toute donnée du cache lorsque les deux sont présentes simultanément en local.

Les structures de données suivantes sont utilisées pour représenter la progression de la requête au niveau du résolveur :

-SNAME : nom de domaine sur lequel porte la recherche.

-STYPE : QTYPE de la requête.

-SCLASS : QCLASS de la requête.

-SLIST : c'est une structure qui décrit les serveurs de noms et la zone concernée par la requête en cours de traitement par le résolveur. Cette structure garde la trace des serveurs de noms que le résolveur estime actuellement être les plus susceptibles de détenir l'information désirée ; cette trace est remise à jour si l'information arrivant au résolveur est de nature à remettre en cause cette estimation. Cette structure contiendra un champ équivalent à un nom de zone, des champs pour représenter les serveurs de noms identifiés pour cette zone, les adresses de ces derniers, et des informations de type "historique" permettant d'estimer quel est le serveur suivant le mieux placé pour obtenir les informations recherchées.

SBELT : c'est une structure "ceinture de sécurité" d'une forme identique à SLIST, initialisée à partir d'un fichier de configuration, et qui liste les serveurs qui devraient être utilisés lorsque le résolveur ne dispose plus ou pas assez d'informations locales pour guider la sélection de serveurs de noms.

-CACHE : c'est une structure qui enregistre les résultats de précédentes réponses. Les résolveurs sont responsables de la purge des RR dont la durée de vie a expiré (dans le cache). La plupart des implémentations convertissent l'intervalle spécifié dans les RR reçus en une sorte de date "absolue" de validité lorsque le RR est enregistré dans le cache. Plutôt que décrémenter les durées de vie individuellement, le résolveur n'a plus qu'à ignorer ou supprimer les RRs non valides lorsqu'il les rencontre au cours d'une recherche, ou encore supprimer les RRs dont la date de validité est antérieure à la date courante lors d'opérations de lecture.

2.3.3 L'algorithme résolveur [1]

Contient quatre étapes :

Etape1.Vérifier si la réponse est disponible en local, si c'est le cas on la renvoie au client.

Etape2.Déterminer les "meilleurs" serveurs à contacter.

Etape3.Leur envoyer des requêtes jusqu'à ce qu'un d'eux réponde.

Etape4.Analyser la réponse :

1-Si la réponse répond à la question posée, ou stipule une erreur de nom, retourner l'information au client et mettre cette information en cache.

2-Si la réponse indique une redirection vers un serveur le plus proche, mettre en cache les références de ce nouveau serveur, et retourner à l'étape 2.

3-Si la réponse est un CNAME tout en n'étant pas la réponse attendue, cacher le CNAME, remplacer le SNAME par le nom canonique lu dans le RR CNAME et retourner à l'étape 1.

4-Si la réponse rend compte d'une défaillance du serveur ou autre cas ininterprétable, supprimer le serveur de la SLIST et retourner à l'étape 3.

2.3.4 Organigramme résolveur

L'algorithme ci avant décrit par la RFC 1034 a été traduit en un organigramme illustré par la figure ci dessous; les quatre étapes y sont mises en valeur.

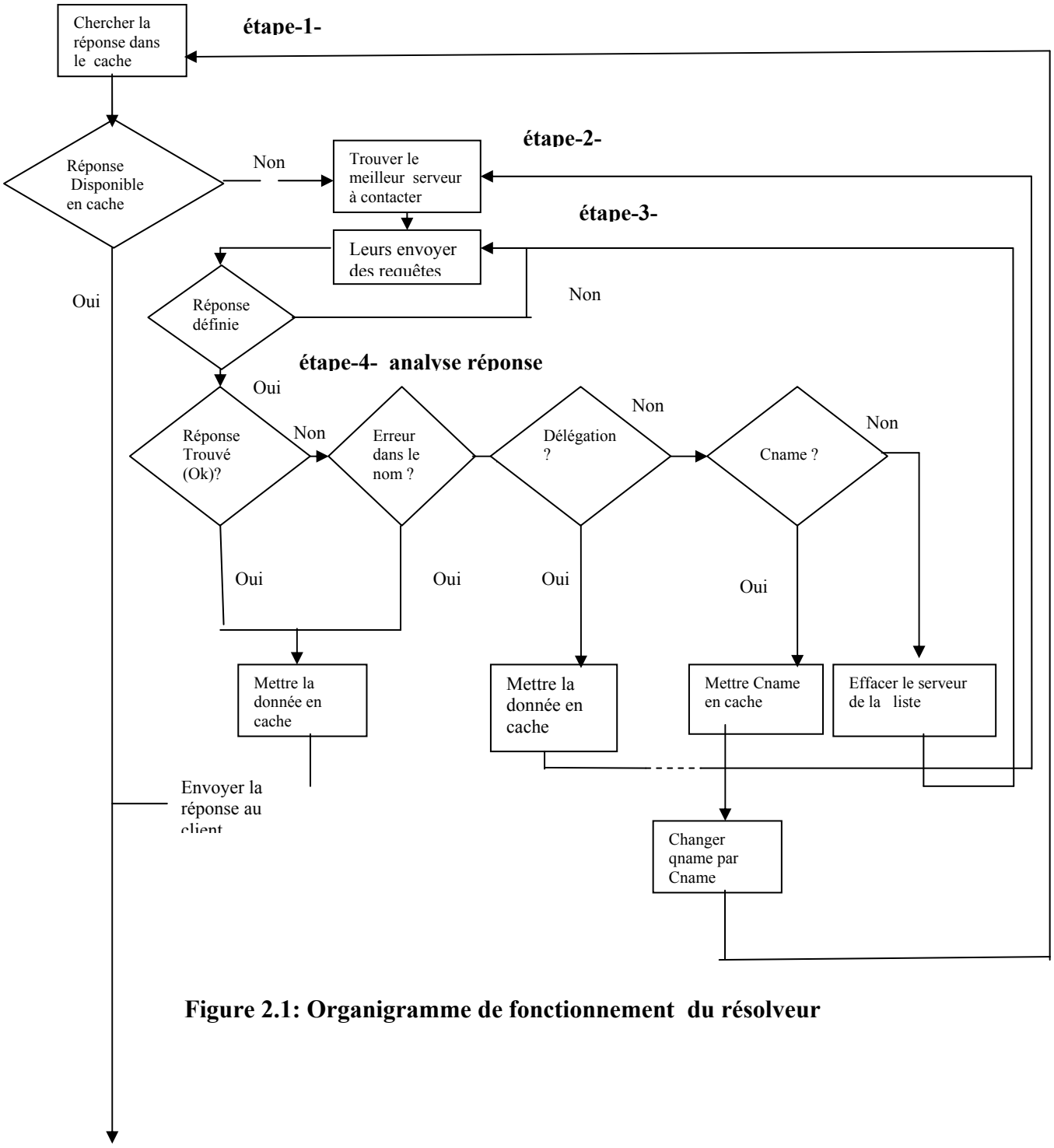


Figure 2.1: Organigramme de fonctionnement du résolveur

2.3.4.1 Etape 1 ou la recherche de la réponse dans le cache

L'étape 1 cherche la donnée désirée dans le cache. Si la donnée est trouvée, on suppose qu'elle est suffisamment fiable pour une exploitation normale. Certains résolveurs disposent d'une option configurable par l'utilisateur qui force à ignorer les données du cache et à consulter directement un serveur faisant autorité. Ce fonctionnement n'est pas recommandé par défaut. Si le résolveur dispose d'un accès direct à la définition de zone d'un serveur de nom local, il cherchera à établir si les données demandées n'y sont pas présentes. Dans l'affirmative, on utilisera de préférence les données d'autorité plutôt que les données trouvées dans le cache.

La figure 2.2 ci après illustre l'étape traitant de la recherche dans le cache.

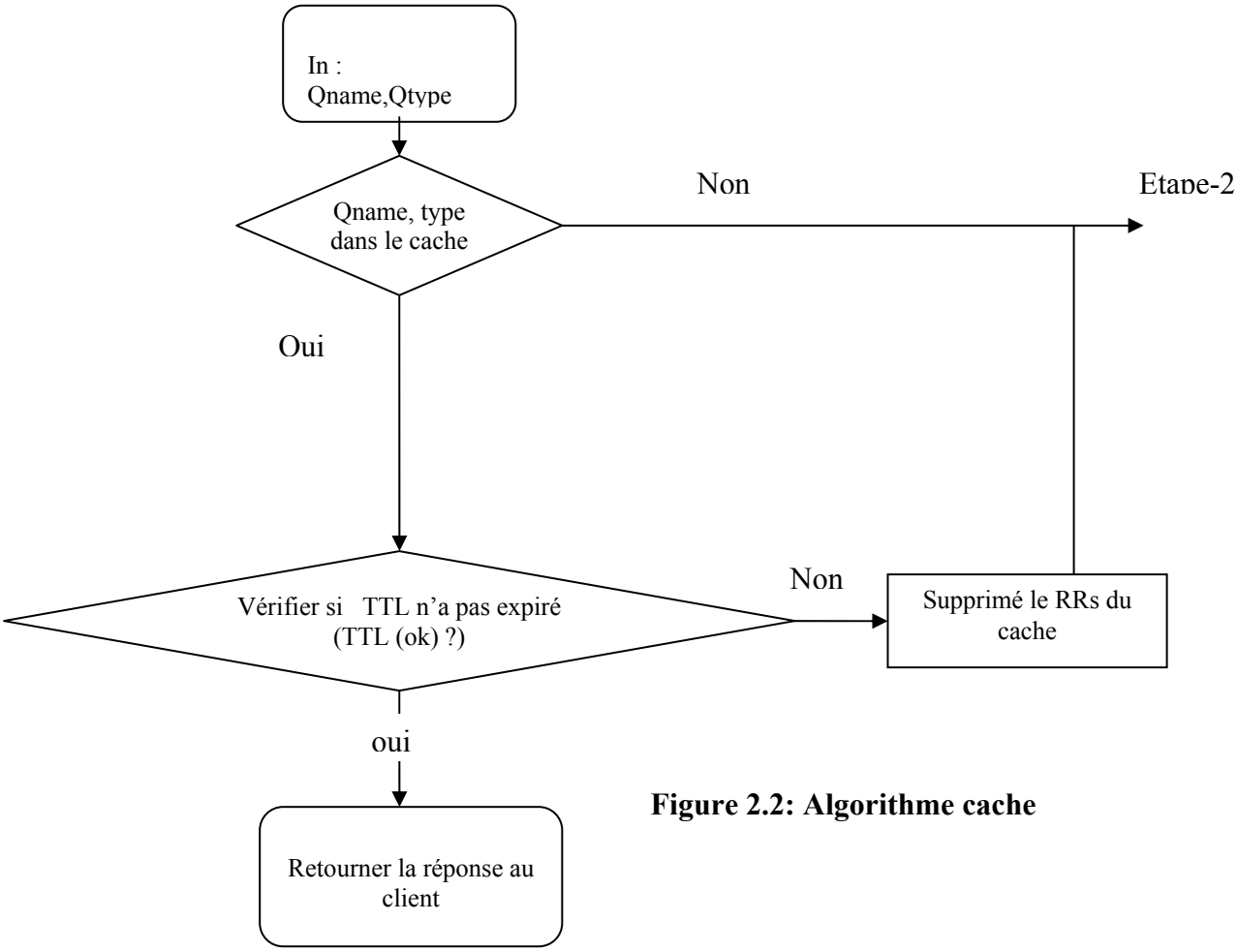


Figure 2.2: Algorithme cache

2.3.4.2 Etape 2 ou localisation du meilleur serveur à contacter

Le but principal de cette partie de l'algorithme est de trouver le serveur le plus proche. Cette étape consiste en la recherche auprès de quel serveur de nom l'on va requérir la donnée recherchée. La stratégie usuelle est de chercher d'abord des RR de serveurs de noms disponibles en local, en partant du domaine SNAME, puis le père de SNAME, son grand-père, et ainsi de suite vers la racine. Ces RRs NS donnent la liste des noms des hôtes de la zone coïncidant avec ou englobant SNAME. On copie ces noms dans SLIST, puis on détermine leur adresse à l'aide des données locales. Les concepteurs de résolveurs devront considérer les priorités dans l'ordre suivant :

1. Donner une réponse à chaque fois que possible.
2. Donner les réponses aussi vite que possible.

Si la recherche de RR NS échoue, alors le résolveur initialisera la SLIST à partir de la structure SBELT. L'idée de base est que, lorsque le résolveur ne sait pas par où commencer pour rechercher une information, il utilisera une liste prédéfinie dans un fichier de configuration de serveurs supposés pouvoir être utiles. On intégrera usuellement dans cette liste deux serveurs sur la racine et deux serveurs dans la zone où réside l'hôte. Le chiffre de deux (minimum) est donné pour assurer la redondance des données. Les serveurs de la racine pourront permettre un accès éventuel à d'autres parties de l'espace de domaines. Les deux serveurs locaux permettront au résolveur de pouvoir être capable de continuer la résolution de noms locaux, même si le réseau local se retrouve isolé d'Internet suite à la défaillance d'une liaison ou d'un routeur.

En plus de fournir les adresses et noms des serveurs, la structure de données SLIST peut être triée de façon à donner une indication sur le meilleur serveur à utiliser, et s'assurer que toutes les adresses et noms de serveurs seront contactés chacun son tour. Le tri peut être une simple fonction de prédominance des adresses locale sur les adresses externes.

Cette étape est illustrée par la figure suivante.

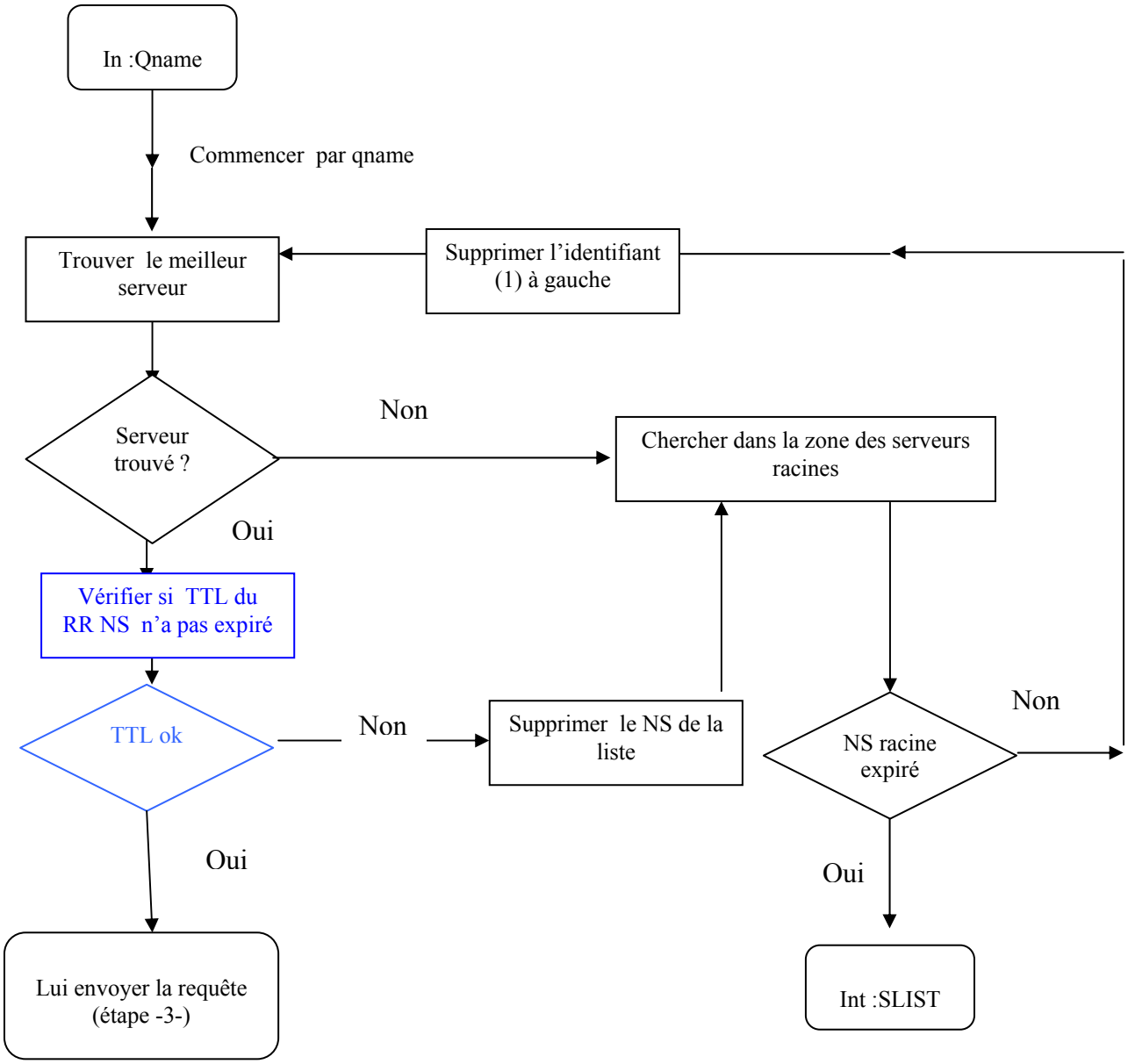


Figure 2.3 : Algorithme recherche du meilleur serveur

2.3.4.3 Etape 3 ou envoie des requêtes jusqu'à obtenir une réponse

Dans l'étape 3, le résolveur émet des requêtes jusqu'à ce qu'une réponse soit trouvée. La stratégie consiste à utiliser toutes les adresses à tour de rôle.

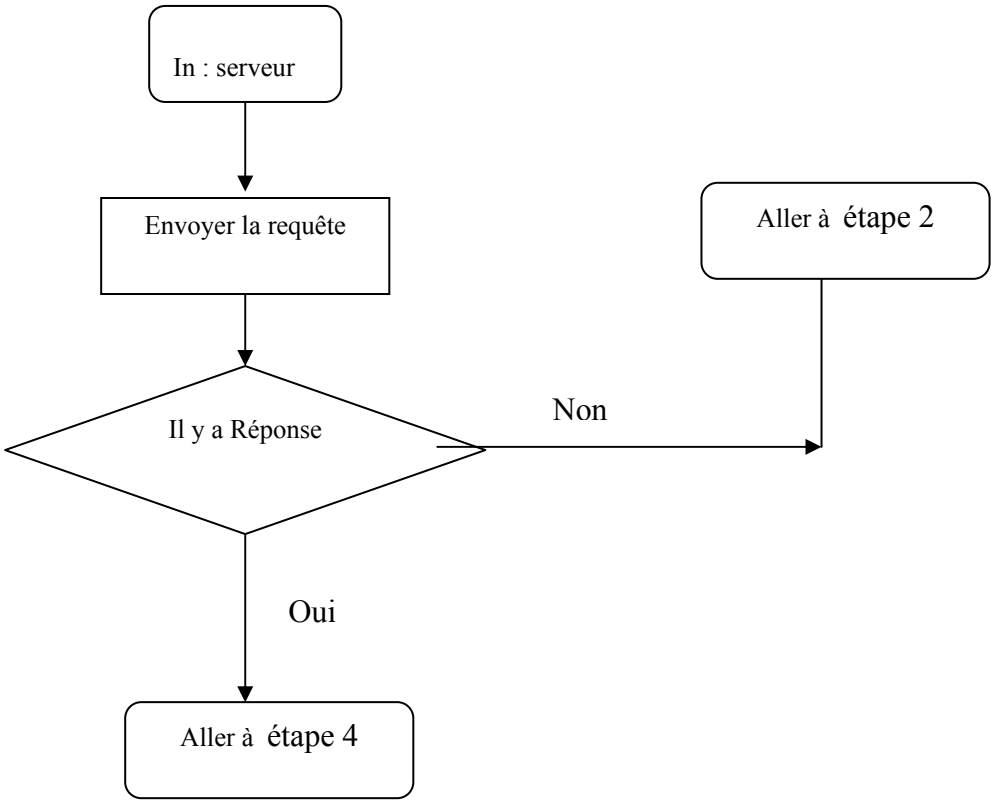


Figure 2.4: Envoyer requête

2.3.4.4 Etape 4 ou analyses des réponses

Le résolveur passe à cette étape quand la section answer n'est pas vide et le qtype des RRs dans cette section est identique à celui de la requête.

Le résolveur devra vérifier que la réponse correspond bien à la requête émise grâce à la lecture du champ ID dans la réponse.

L'étape 4 s'occupe donc de l'analyse des réponses. Le résolveur devra faire preuve d'une grande rigueur dans l'analyse de ces réponses. Il devra en outre déterminer que la réponse correspond bien à la requête émise grâce à la lecture du champ ID dans la réponse. La réponse idéale proviendrait d'un serveur autoritaire donnant les informations (en général l'adresse) attendues ou encore une erreur de nom.

L'information est alors transmise au client et recopiée dans le cache en prévision d'une requête future identique si sa durée de vie (TTL) est supérieure à zéro.

2.3.4.4.1 Etape 4a, b ou cas d'une réponse trouvée

Si la réponse correspond à la question posée, le nom de domaine sur lequel porte la question est trouvée ou stipule une erreur de nom. Retourner l'information au client et mettre cette information en cache.

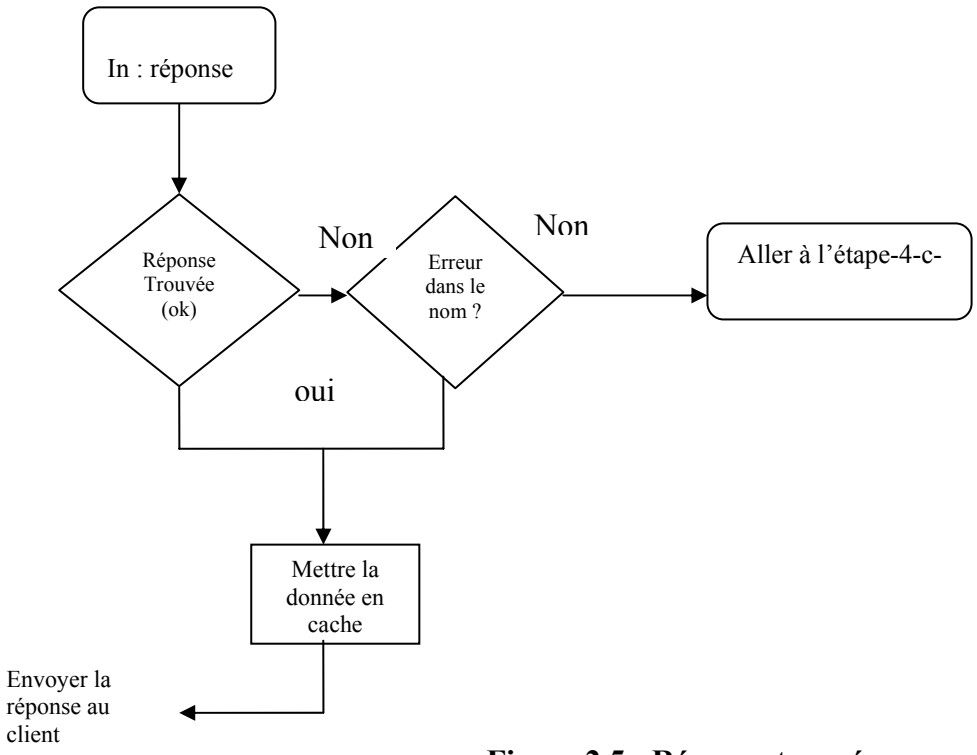


Figure 2.5 : Réponse trouvée

2.3.4.4.2 Etape 4c ou traitement de la délégation

Au sein d'un domaine, on choisit souvent de transférer la responsabilité de certains sous ensembles de noms; c'est ce qu'on appelle une délégation et cela a pour conséquence la création d'une nouvelle zone dite zone fille de la première (zone parente).

*Si la réponse indique une redirection à effectuer, le résolveur vérifiera d'abord si le serveur mentionné est plus proche de la réponse que les serveurs dont on dispose dans SLIST. Ceci sera fait en comparant le décompte d'identifiants correspondants

dans SLIST avec celui de SNAME dans le RR NS du message de redirection. Si tel n'est pas le cas, la réponse est considérée comme dénuée d'intérêt et sera ignorée. Si la redirection est utilisable, le RR NS de redirection et toute RR d'adresse obtenue pour les serveurs délégués devront être copiés dans le cache. Les serveurs de noms sont ajoutés à SLIST, et la recherche recommence

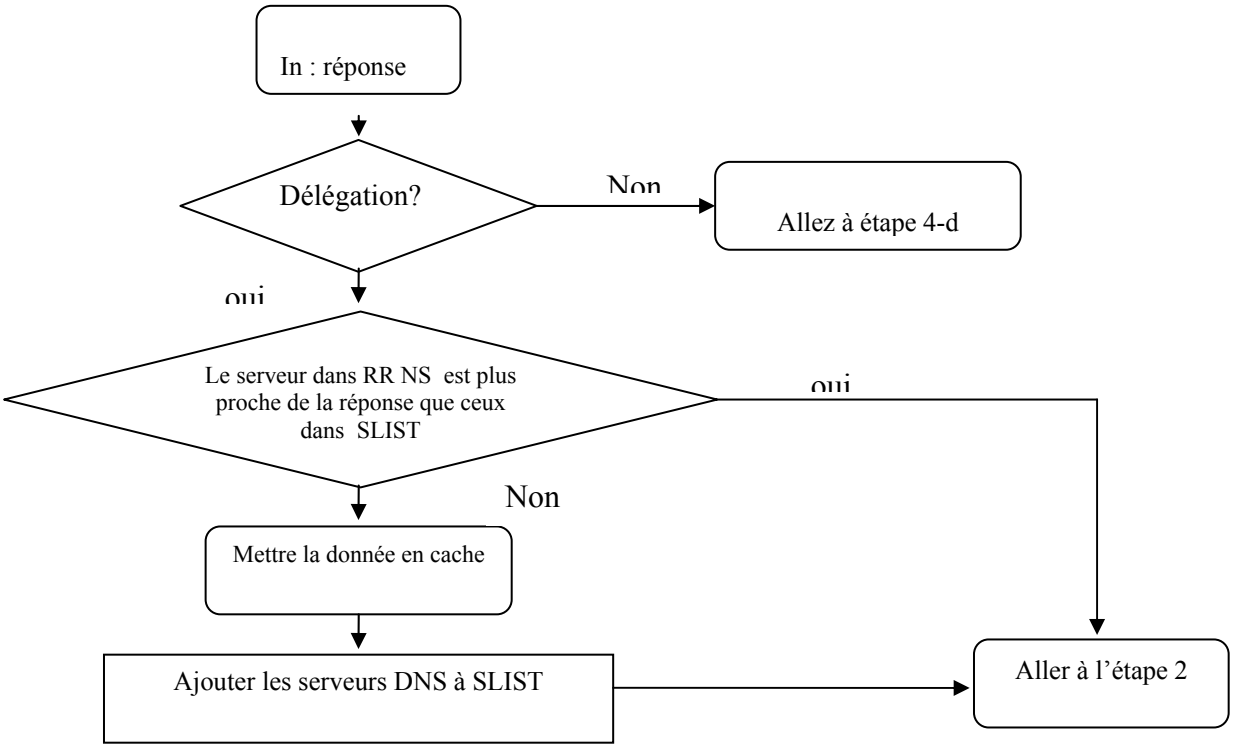


Figure 2.6 : Délégation

2.3.4.4.3 Etape 4d ou analyse de Cname

Lorsqu'il essaie de résoudre une requête particulière, le résolveur peut s'apercevoir que le nom demandé est un alias. Par exemple, le résolveur s'aperçoit que le résultat d'une translation est de ce type lorsque le RR renvoyé est un CNAME. Dans la plupart des cas, un résolveur relancera la résolution sur le nouveau nom translaté donné dans le RR CNAME.

*Si la réponse contient un CNAME, la recherche doit être réitérée avec le nom canonique CNAME sauf si la réponse donnée contient déjà l'information pour ce même nom canonique, où si la requête originale portait effectivement sur un CNAME lui-même.

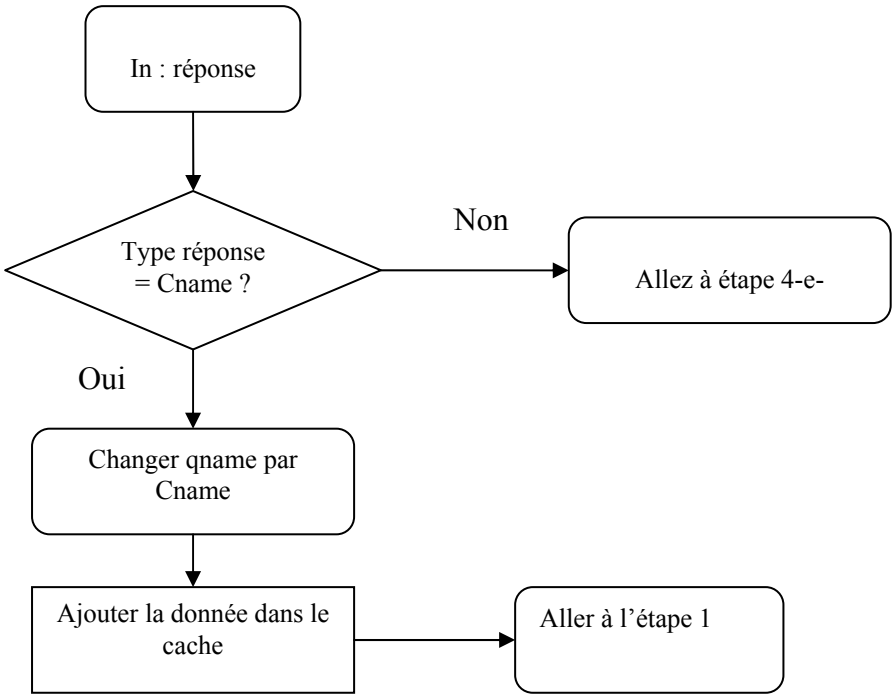


Figure 2.7: Réponse CNAME

2.3.4.4.4 Cas défaillance du serveur

Si la réponse rend compte d'une défaillance du serveur ou autre cas bizarre, on supprime le serveur de la SLIST et on retourne à l'étape 3.

Cette étape est illustrée ci après.

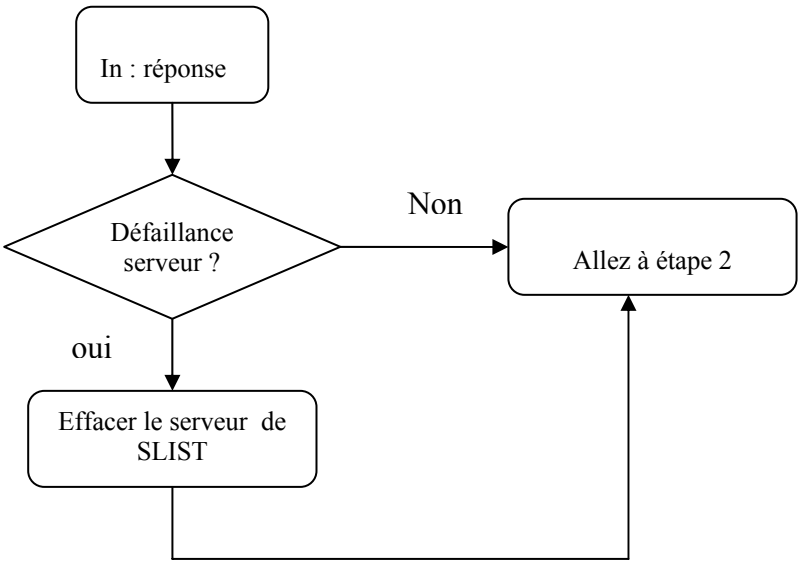


Figure 2.8: Défaillance serveur

2.3.5 Le serveur de nom

La principale activité d'un serveur de noms est de répondre aux requêtes standard. La requête et sa réponse sont toutes deux véhiculées par un message de format standardisé décrit dans la RFC 1035. Pour rappel, la requête contient des champs QTYPE, QCLASS, et QNAME.

La manière dont le serveur de noms répond peut être différente selon que le serveur utilise le mode récursif ou non :

- Le mode de fonctionnement du point de vue serveur est le mode non récursif, dans la mesure où le serveur peut répondre à la requête uniquement sur la base de ses informations locales : la réponse contient une erreur, la réponse demandée, ou donne la référence à un autre serveur plus susceptible de disposer de l'information demandée. Tous les serveurs de noms se doivent d'implémenter le mode non récursif.
- Le mode le plus simple de fonctionnement du point de vue du client est le mode récursif, dans la mesure où dans ce mode, le serveur prend à son tour le rôle de résolveur et ne peut retourner qu'un message d'erreur ou une réponse valide, mais jamais de référence. Ce service est optionnel dans un serveur de noms; ce dernier pouvant de plus choisir de restreindre les possibilités de clients gérant le mode récursif.

  Le service récursif est utile dans plusieurs situations :

- une implémentation simplifiée d'un résolveur qui ne sait exploiter d'autres réponses qu'une réponse directe à la question.
- un réseau dans lequel intervient une politique de cache commun plutôt qu'un cache individuel par client.

Le service non récursif est approprié si le résolveur est capable de façon autonome de poursuivre sa recherche et est capable d'exploiter l'information supplémentaire qui lui est envoyée pour l'aider à résoudre son problème.

Notons que le serveur de noms ne doit pas utiliser le service récursif s'il n'a pas été explicitement demandé par le client. Lorsque le service récursif est demandé et est disponible, la réponse récursive à une requête doit être l'une des suivantes :

- La réponse à la requête, éventuellement préfacée par un ou plusieurs RR CNAME indiquant les alias trouvés pendant la recherche de la réponse.
- Une erreur de nom indiquant que le nom demandé n'existe pas. Celle-ci peut inclure des RR CNAME qui indiquent que la requête originale pointait l'alias d'un nom qui n'existe pas.

Si le service récursif n'est pas requis, ou n'est pas disponible, la réponse non récursive devra être l'une des suivantes :

- Une réponse d'erreur d'autorité indiquant que le nom n'existe pas.
- Une combinaison :

* des RR qui répondent à la question, avec indication si les données sont extraites d'une zone ou d'un cache.

* d'une référence à un serveur de noms qui gère une zone plus proche du nom demandé que le serveur qui a été contacté.

- Les RRs que le serveur de nom pense être utile au requérant pour continuer sa recherche.

2.3.5.1 Algorithme serveur DNS [1]

L'algorithme du serveur contient six étapes :

1. Marquer ou effacer la valeur de "récursion admissible" (RA) dans la réponse suivant que le serveur de noms souhaite proposer le service récursif ou non. Si le service récursif est disponible et requis par le marquage du bit RD dans la requête, aller à l'étape 5, autrement continuez à l'étape 2.

2. Chercher, dans les zones disponibles, celle qui est l'ancêtre le plus proche de QNAME. Si une telle zone existe, aller à l'étape 3, sinon sautez à l'étape 4.

3. Commencer la recherche dans la zone. Le processus de recherche peut s'achever de plusieurs manières :

a. Si le nom QNAME est trouvé entièrement, le noeud a été trouvé.

Si l'information présente dans le noeud est un CNAME et QTYPE ne correspond pas à CNAME, copier le RR CNAME dans la section "réponse" du message retourné, changer QNAME en le nom canonique dans le RR CNAME, et retournez à l'étape 1. Autrement, copier tous les RR qui correspondent au QTYPE dans la section "réponse" et sauter à l'étape 6.

b. Si la recherche nous amène hors des données "autorisées", il s'agit d'une référence. Ceci arrive lorsque nous rencontrons un noeud contenant des RRs NS indiquant que nous

arrivons à un point de coupe vers une sous-zone.

Copier les RR NS de la sous-zone dans la section "autorisée" de la réponse. Mettre toute adresse disponible pouvant aider à atteindre la sous-zone dans la section additionnelle, en utilisant des RR "glue" si ces adresses ne peuvent être extraites que d'une partie non autorisée ou du cache. Sauter à l'étape 4.

c. Si pour l'un des identifiants du nom, aucune correspondance n'est possible (c à d. l'identifiant correspondant n'existe pas), voir si un identifiant "*" [1] existe.

Si l'identifiant "*" n'existe pas, vérifier si le nom que nous cherchons est le QNAME original de la requête et non un nom donné par une ou plusieurs redirections dans des CNAME. Si le nom est l'original, préparer une réponse d'erreur "autorisée" et sortir.

Sinon on sort de l'algorithme sans autre action. Si l'identifiant "*" existe, chercher les RRs de ce noeud correspondant au QTYPE de la requête. S'il en existe, les copier dans la section "réponse", mais en requalifiant le propriétaire (owner) des RR comme étant QNAME, et non celui du RR contenant l'identifiant "*". Sauter en 6.

4. Commencer la recherche dans le cache. Si QNAME y est trouvé, copier dans la section "réponse" tous les RRs qui y sont rattachés et dont le QTYPE est conforme. S'il n'existe pas de délégation sur ces données pour ce qui concerne "l'autorisation", chercher dans le cache la meilleure information d'autorisation, et la placer dans la section "autorisation". Aller en 6.

5. Utiliser le résolveur local pour répondre à la requête. Enregistrer les résultats, y compris tout CNAME intermédiaires, dans la section réponse.

6. Sur la base des données locales uniquement, tenter d'ajouter tous les autres RRs qui pourraient être utiles dans la section additionnelle de la réponse et sortir.

2.3.5.2 Organigramme serveur DNS

L'organigramme suivant résume les différentes étapes décrites dans l'algorithme du serveur DNS décrit ci-dessus.

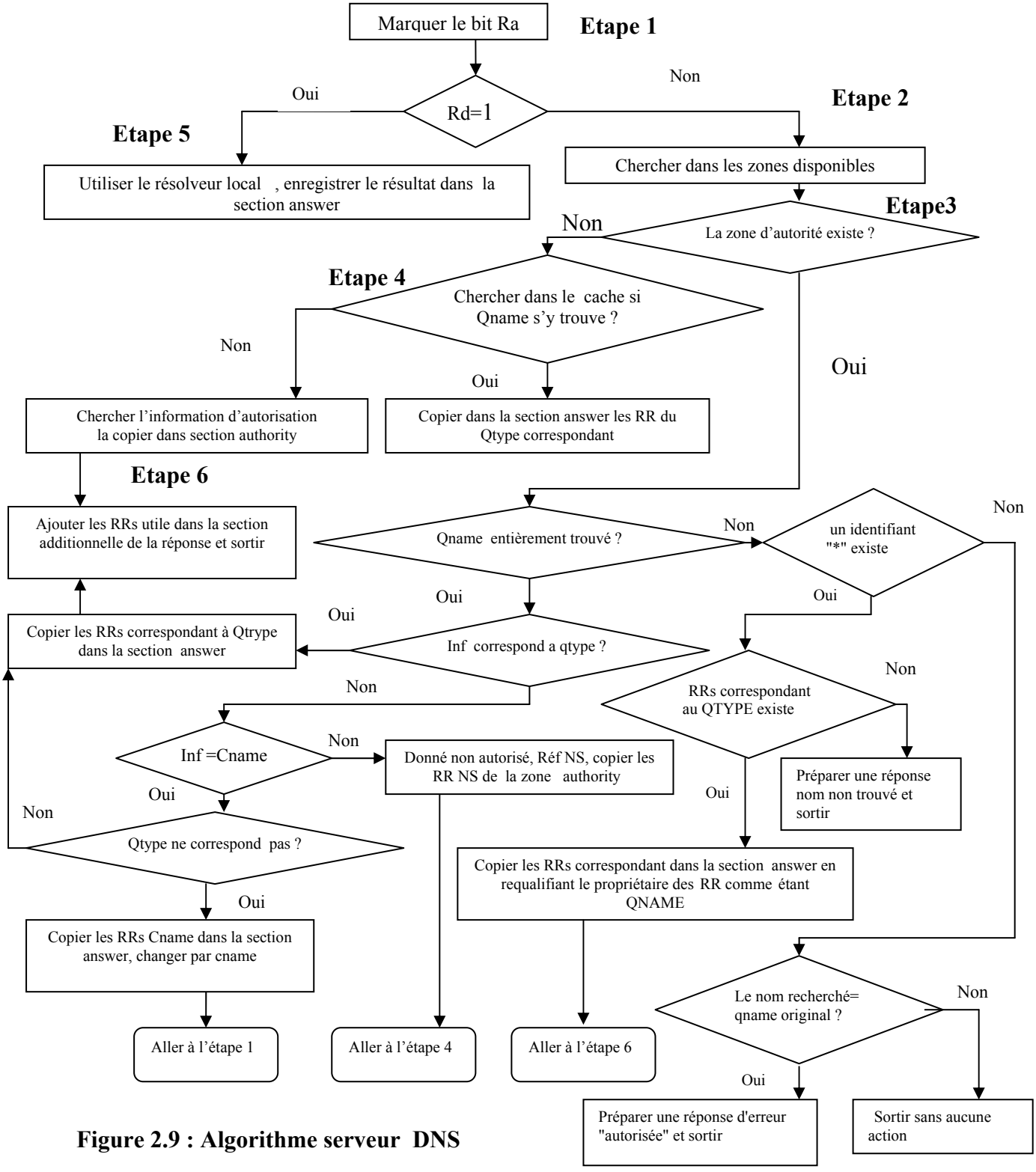


Figure 2.9 : Algorithme serveur DNS

2.4 Conclusion

L'implémentation d'une résolution de noms passe au préalable par l'étude des algorithmes normalisés de fonctionnement des constituants du système de résolution. Comme on vient de le voir, deux constituants majeurs apparaissent; le résolveur et le serveur proprement dit.

L'étude des algorithmes respectifs a conduit à la déduction d'organigrammes qui nous seront nécessaires lors de la phase d'implémentation.

Penser à l'implémentation d'un serveur DNS passe au préalable par la détermination des fonctions à implémenter. Ce qui conduit à trois formes d'implémentation possible : un résolveur, un serveur récursif sans gestion de zones locales, un serveur récursif avec des gestions de zones locales et serveur autoritaire (ne faisant gérer que les données de zones locales).

Nous verrons dans le chapitre suivant l'approche d'implémentation suivi tenant compte de l'orientation que nous nous sommes fixés.

3 Méthodologie de développement d'un serveur DNS en VHDL

3.1 Introduction

L'objectif est l'implémentation hardware du service DNS, dont le fonctionnement est décrit et illustré sous forme d'organigrammes dans le chapitre précèdent. L'intérêt est son confinement pour une meilleure fiabilité et plus grande sécurisation.

Le présent chapitre traite de la méthodologie de développement suivie.
Nous dresserons dans une première partie de ce chapitre, une introduction au diagramme en Y introduit par Gajski décrivant les trois vues d'une architecture.
En effet, notre approche de conception repose sur ce modèle.

La deuxième partie sera quant à elle consacrée à la méthodologie que nous proposant pour l'implémentation hardware du service DNS, en utilisant le langage VHDL pour la modélisation. Nous détaillerons par la suite les différentes étapes du flot de conception suivi.

3.2 Méthodologie de conception

La méthode de conception traditionnelle de composants consistait tout d'abord à transformer un cahier des charges définissant les fonctionnalités du système en spécifications. Ces spécifications sont alors utilisées pour générer l'architecture du système en terme d'interconnexions de blocs fonctionnels.

La conception de circuits de grande complexité suppose de partir de spécifications abstraites. Il est en effet plus aisé pour un concepteur, de manipuler des fonctions que de dessiner transistor par transistor le schéma de son circuit. Idéalement, on souhaite fournir aux outils de conception des algorithmes exprimés de manière fonctionnelle, et en obtenir le plan de masse des circuits.

3.2.1 Expression d'une architecture

Un circuit est destiné à résoudre un problème. On peut donc voir ce circuit d'une manière plus ou moins proche de la réalité physique ou de l'aspect algorithmique.

On distingue trois angles de vue (plus couramment désignés par les termes domaines d'expression pour exprimer une architecture (figure 3.1).

Ces domaines d'expression sont eux-mêmes découpés en cinq niveaux de description plus ou moins abstraits par rapport aux détails du circuit.

Les trois domaines d'expression sont:

- le domaine comportemental, qui décrit le comportement d'un circuit, autrement dit ce qu'il fait ;
- le domaine structurel, qui voit le circuit comme une interconnexion d'objets architecturaux de base, et fait le lien entre le domaine comportemental et le domaine physique ;
- le domaine physique, souvent représenté par le dessin de masque du circuit, et qui présente la structure de la réalisation physique de l'architecture.

La spécification de départ d'un circuit relève généralement du domaine comportemental. Diverses étapes de synthèse mènent ensuite à une description physique.

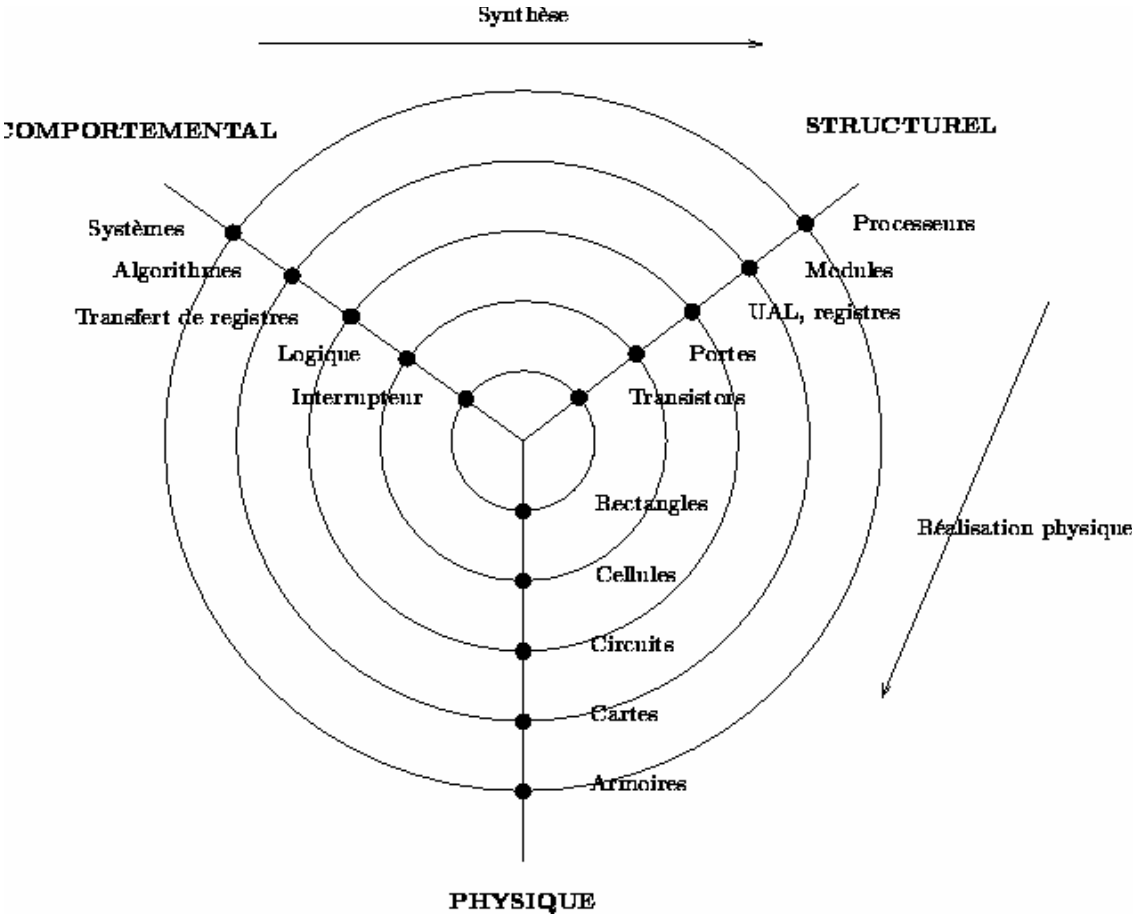


Figure 3.1 : Diagramme en Y

3.2.2 Spécification de systèmes

Les problèmes posés par la conception de systèmes sont liés aux quelques notions suivantes.

- Large choix de technologies. Le concepteur de systèmes doit faire face à une palette de choix de conception en constante augmentation (technologies de fabrication, méthode de conception, choix architecturaux).
- Partage matériel/logiciel. Les systèmes comportent une part de plus en plus significative de logiciel, en raison de la disponibilité de cœurs de processeurs (RISC, DSP, microcontrôleurs), comme éléments de bibliothèque. Il s'agit de pouvoir réaliser une

spécification indifféremment sous forme matérielle ou logicielle, de pouvoir déterminer le partitionnement et d'engendrer automatiquement des interfaces.

- Complexité des systèmes. La complexité croissante des systèmes est une conséquence directe de l'évolution des technologies cibles vers des niveaux d'intégration plus élevés.

  La spécification d'un système est généralement réalisée au niveau comportemental. Autrement dit le comportement de l'application est exprimé par des fonctions. Ces fonctions décrivent le calcul à réaliser et les relations entre entrées et sorties.

  Différents niveaux d'abstraction peuvent être répertoriés : interrupteur, logique, transfert de registres, algorithmique et système.

3.2.3 Synthèse d'architectures

La synthèse est un procédé automatisé permettant de faire passer la représentation d'une architecture du domaine comportemental au domaine structurel. On parle aussi de synthèse physique pour désigner le passage du domaine structurel au domaine physique.

La synthèse architecturale désigne, quant à elle, le processus automatisé qui génère à partir d'une spécification exprimée dans le domaine comportemental la description d'un circuit en terme de blocs interconnectés, qu'il passe directement ou non du domaine comportemental au domaine structurel.

3.3 Approche de conception

Notre approche de conception, déduite des considérations qui viennent d'être introduite et de l'outil de conception que nous avons utilisé (Xilinx ISE), est illustrée par la figure 3.2.

L'approche envisagée pour le passage des spécifications abstraites vers l'implémentation sur circuit du serveur DNS consiste en le passage d'une modélisation en langage de description matériel VHDL vers une implémentation sur circuit FPGA.

On s'intéresse plus particulièrement à la méthodologie de développement qui aboutit à la définition d'un modèle d'exécution de l'application DNS. Nous avons utilisé dans notre

approche le langage VHDL et nous présentons dans ce qui suit, les diverses phases de l'approche envisagée.

3.4 Méthodologie de développement d'un serveur DNS en VHDL.

Les phases de l'approche de conception que nous proposons sont résumées comme suit :

1-Etude des algorithmes proposés par P.Mockapetris, dans la RFC 1034 et les spécifications d'implémentation du protocole DNS dans la RFC 1035.

2-Translation des algorithmes en organigrammes de fonctionnement détaillant les différentes étapes de résolution.

3-Définition du modèle architectural permettant l'implémentation du DNS sur circuit.

4- Définition du flot de conception du serveur DNS en VHDL.

Les étapes de la méthodologie suivie seront présentées dans les sections suivantes. En particulier le flot de conception.

3.4.1 Choix du Langage

Avec l'apparition des outils de synthèse et des langages de description du matériel, la méthodologie de conception a dû évoluer. Le système est décrit à un niveau d'abstraction de plus en plus élevé à l'aide des langages de description du matériel.

Le langage VHDL est un langage de description matériel (HDL) qui permet de travailler avec un niveau d'abstraction élevé. Il représente un outils de base pour la conception de systèmes logiques intégrés câblés, que le produit final soit construit sur des composants électriquement configurables FPGA ou des circuits intégrés spécifiques ASIC.

Le langage VHDL supporte aussi les concepts de parallélisme et de communication interprocessus. Plusieurs approches utilisent ce langage pour décrire les parties matérielles d'un système en conjonction avec un langage logiciel (C) pour décrire les parties logicielles. Nous avons choisi ce langage car dans notre cas le langage VHDL est le mieux adapté, puisque il s'agit dans notre approche d'un système n'incluant que des parties matérielles, et aussi pour les raisons suivantes :

VHDL est un langage qui :

- Offre quelques particularités facilitant la réutilisation de composants matériels, comme la généricité.
- décrit la fonctionnalité souhaitée et ceci indépendamment de la technologie.
- Permet d'écrire la spécification d'un système (description comportementale ou fonctionnelle)
- Permet de construire par interconnexion des modules élémentaires (description structurelle ou physique)
- Permet de valider par simulation les différents types de description. Une fois le codage terminé, la simulation fonctionnelle permet une première vérification du système.
- Permet l'utilisation de programmes de synthèse automatique.

Pour toutes ces raisons, VHDL se présente comme étant le mieux adapté et le plus pratique, pour nos spécifications système et la modélisation, puis l'implémentation du service DNS sur un circuit en vu de sa sécurisation.

Il faut noter que le langage de description matériel VHDL est aussi intéressant pour la facilité de modification et de réutilisation du modèle précédent pour un nouveau modèle, ceci donne la possibilité d'optimiser le modèle déjà conçu .

3.4.2 Les étapes de la méthodologie

3.4.2.1 Etude des algorithmes

La première étape est celle de l'étude des algorithmes du résolveur et du serveur DNS et des spécifications du protocole DNS tel qu'il, sont décrits, respectivement dans les références RFC 1034 et 1035.

3.4.2.2 Translation des algorithmes en organigrammes fonctionnement

La seconde étape est la translation de ces algorithmes en organigrammes de fonctionnement. Cela consiste à proposer des organigrammes qui illustrent les étapes de fonctionnement, du résolveur et du serveur. Dans cette phase nous donnerons les descriptions des déférentes étrapes représentées dans les organigrammes proposés.

3.4.2.3 Définition du modèle architectural

L'étape suivante est la définition d'un model architectural, qui divise le système à implémenter en sous systèmes .Cette phase est une phase essentielle dans notre méthodologie, puisque elle permet de transformer la spécification initiale en une solution d'architecture désirée. C'est à ce niveau que la décision concernant l'architecture du circuit permettant l'implémentation du serveur DNS se fait.

Comme il va être décrit dans le chapitre 4, la solution architecturale proposée, est constituée d'un ensemble de blocs fonctionnels communicants.

3.4.2.4 Flot de conception

La dernière étape consistera donc à définir le flot de conception du serveur DNS en VHDL. Cette étape est détaillée si après.

La Figure 3.2 présente les différentes étapes du flot de conception, allant de l'étude des algorithmes et des spécifications jusqu'à la synthèse

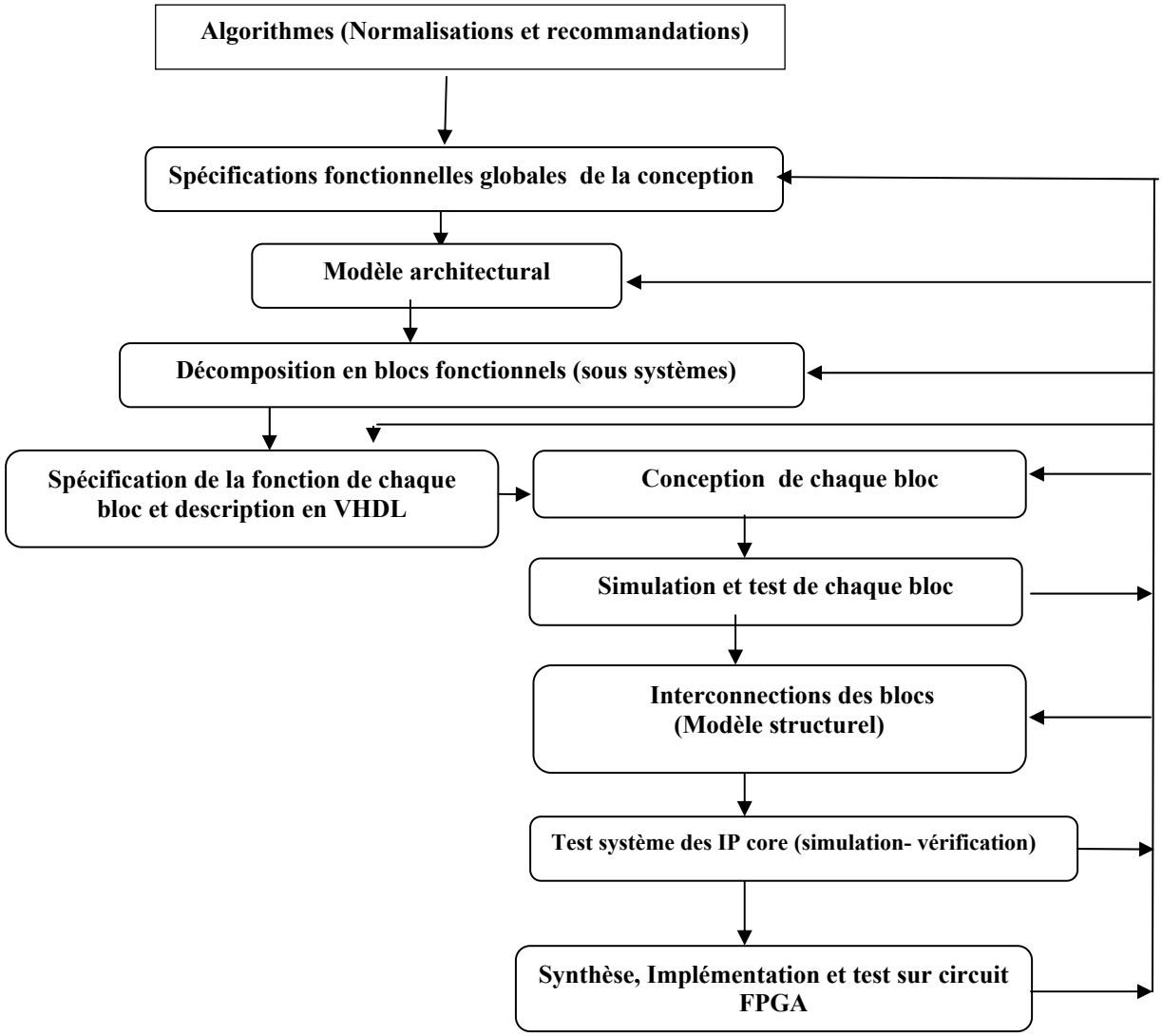


Figure 3.2: Flot de conception du serveur DNS en VHDL

Le flot part d'une spécification système présentant les détails des opérations à effectuer. Cette étape décrit ce que devra réaliser le système. Ces spécifications fonctionnelles seront ensuite transformées en un modèle architectural constitué d'un ensemble de blocs fonctionnels communiquant, afin de fixer la solution architecturale qui sera réalisée.

Suit alors la conception de cette solution. Pour ce faire, chaque bloc fonctionnel est décrit en langage VHDL après avoir spécifié sa fonction dans l'ensemble (les

spécifications de chaque bloc, ainsi que la fonction qu'il réalise seront détaillées dans le chapitre suivant).

Chaque bloc est composé d'un nombre varié de composant selon sa fonction. Ces sous systèmes seront ensuite simulés et testés. Chaque bloc est donc conçu de façon indépendante du test des communications.
Après avoir décrit, simulé et validé les différents blocs constituant notre circuit, on peut passer au modèle structurel. Cette étape consiste à interconnecter les blocs fonctionnels, et donc à rassembler les sous systèmes, afin de réaliser la fonction désirée c-à-d le serveur DNS.
L'avant dernière étape du flot de conception, est la simulation du modèle proposé dans sa forme structurel.
Enfin après synthèse et validation des IP core conçus, l'étape d'implémentation est obtenue automatiquement par l'utilisation d'outils d'implémentation qui permettent la génération des HDL Netlist pour la description des composants et leurs interconnections. Puis la génération du fichier de configuration Bitstream pour charger le système sur la carte FPGA.

3.5 Conclusion

La méthodologie que nous proposons pour l'implémentation du service DNS en utilisant le langage VHDL est développée dans le but d'envisager une approche permettant l'implémentation du service DNS sur circuit. Une implémentation typiquement hardware, dont l'intérêt est de permettre l'isolement de ce service. L'objectif est la sécurisation du DNS qui est une ressource critique et nécessaire au bon fonctionnement des applications Internet.

4 Modélisation et implémentation VHDL d'un serveur DNS autoritaire

4.1 Introduction

Le présent chapitre a trait à l'implémentation d'un serveur DNS autoritaire basée sur les algorithmes de fonctionnement tels que décrits par les RFC 1034 et 1035. Des résultats détaillés pour chaque partie y seront présentés.

La solution architecturale proposée, pour l'implémentation du service DNS en VHDL, permet un isolement physique de ce service. Ainsi, le service DNS s'exécutant sur un circuit est isolé des autres applications tant au point de vue ressources internes que du réseau.

L'architecture qui sera décrite dans ce chapitre est constituée d'un ensemble de blocs fonctionnels communicants, chacun d'eux est décrit à l'aide d'un langage de description
(VHDL) après une spécification préalable de sa fonction. Nous présenterons en particulier le comportement de notre système ainsi que les spécifications de chaque bloc et la fonction qu'il réalise.

Nous commencerons par d´écrire les quatre blocs principaux de notre conception, à savoir le bloc *reception*, le bloc *emission*, le bloc *resolution*, le bloc *configuration*. Pour chacun d'eux, des détails de conception seront présentés, ainsi que leurs architectures et les principaux modules qui les composent. Ensuite, nous présentons les résultats de simulation obtenus pour chaque module conçu séparément ainsi que les résultats de leurs interconnections. Tous les résultats de simulation présentés dans ce chapitre ont été obtenus à l'aide de *Modelsim_6.0.* L'environnement de développement ISE 7.1 fourni par Xilinx [13] a été utilisé pour la saisie et la synthèse de nos descriptions.

4.2Modèle comportemental

L'implémentation du protocole DNS est basée sur trois processus principaux : le processus de réception, celui de l'émission et le processus de résolution .Chaque processus exécute un nombre de fonctions bien déterminé afin de gérer le *message_dns_in* en entrée ou *message_dns_out* en sortie.

Dans les deux processus de réception et d'émission (désassemblage et assemblage des différents champs) le traitement est basé sur la structure du message DNS décrite dans le chapitre 1.

Le processus de réception : Consiste après la réception du message en entrée, en un désassemblage des différents champs constituant le message DNS ; en particulier ceux de l'entête DNS. Chaque champ est caractérisé par sa taille (le nombre de bit qu'il contient) et le type d'information qu'il véhicule .Ces informations seront envoyées au bloc résolution pour le traitement.

Le processus d'émission : Ce processus est l'inverse du processus de réception. Sa fonction principale est de générer les différents champs constituant la réponse pour composer ainsi le message DNS en sortie. Ces différents champs représentent les résultats de la résolution. Ils sont donc fournis par le processus de résolution. Le message de réponse ainsi composé sera envoyé au client via le bloc *emission*.

Le processus de résolution : Il représente la fonction essentielle de notre modèle.

Il est en relation avec les deux autres processus précédemment cités. Le processus de réception qui, après décomposition du message reçu, transmet les informations nécessaires au processus de résolution pour le traitement. Le traitement à ce niveau est essentiellement l'assignation d'une adresse IP aux hôtes réseaux. La réponse générée sera quant à elle transmise au bloc *emission* qui se chargera de la génération du message de sortie.

Le modèle comportemental de notre modèle est illustré par L'organigramme si après.

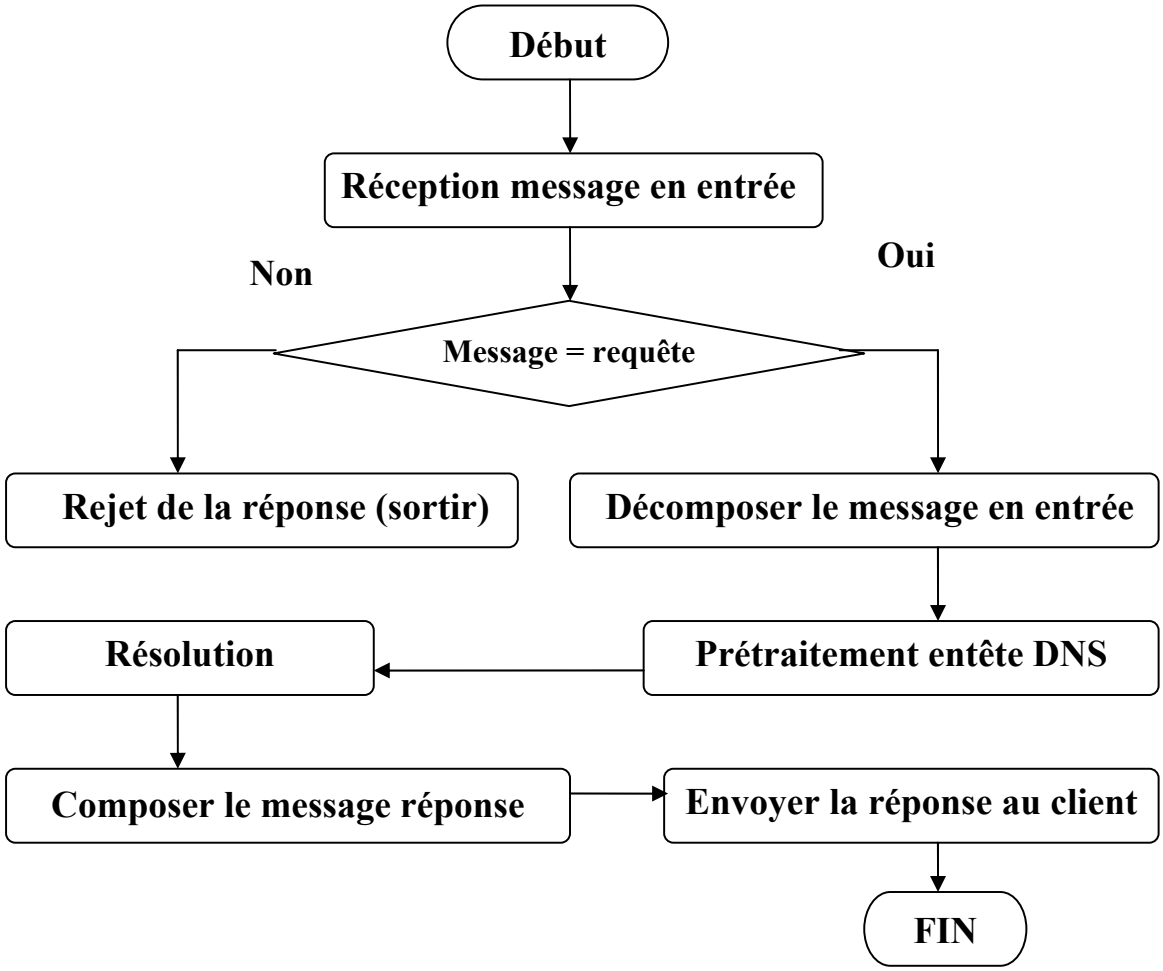


Figure 4.1: Le modèle comportemental du serveur DNS autoritaire

4.3 Modèle structurel

Dans cette section nous allons présenter le modèle architectural proposé, pour l'implémentation du service DNS en VHDL.

Notre architecture est modulaire et est constituée d'un ensemble de blocs fonctionnels communicants.

Chaque bloc représentant un sous système est composé d'un nombre varié de modules selon le rôle qu'il joue dans le système. Ces modules et leurs interconnections au niveau sous système sont décrits en langage VHDL après avoir spécifié au préalable leurs fonctions dans l'ensemble. Les descriptions des différents blocs de notre architecture font chacune l'objet d'une simulation séparée indépendamment du test de communication.
Le schéma de l'architecture globale de notre système est illustré par la figure 4.2-a
Elle représente le diagramme bloc de notre circuit DNS.

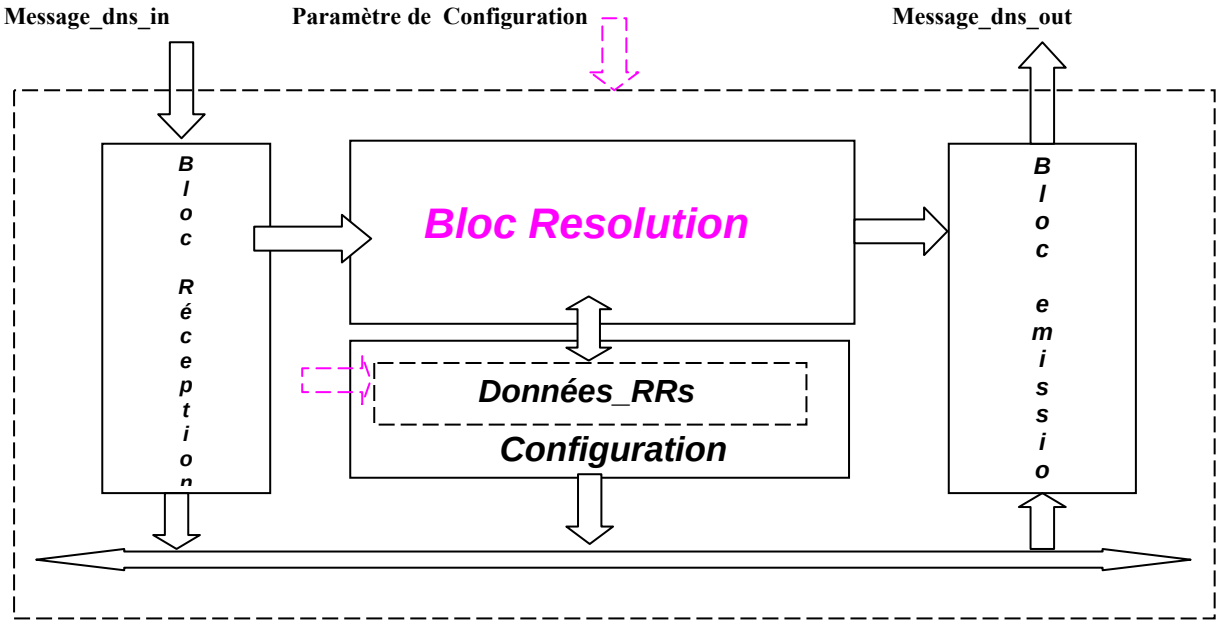


Figure 4.2.a: Diagramme bloc du circuit DNS

La figure 4.2.b quant à elle illustre la vue interne de notre architecture. Sur ce schéma, les signaux d'entrées sorties principaux ont été représentés. Les autres signaux ont été volontairement omis pour ne pas encombrer la figure. Comme montré dans cette figure, l'architecture de notre circuit est divisée en quatre blocs, chacun jouant un rôle bien déterminé.

Notons que mis à part le bloc *resolution*, tous les autres blocs communiquent avec l'environnement extérieur.

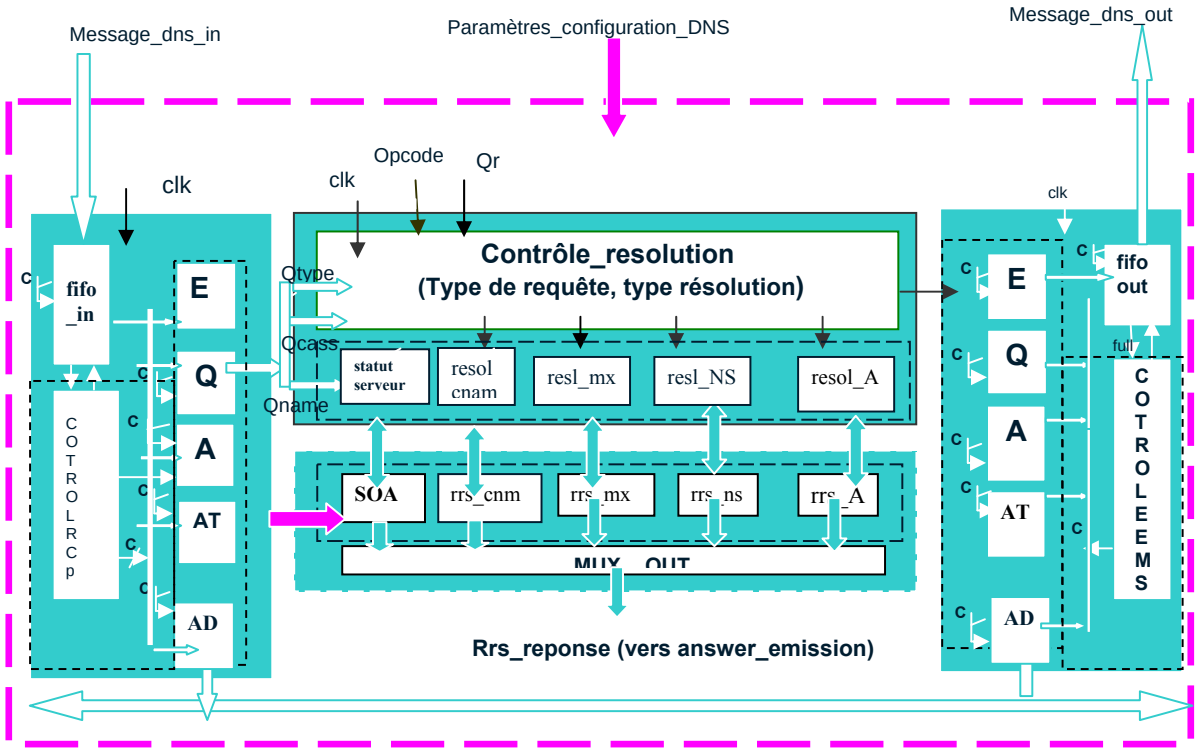


Figure 4.2.b: Architecture interne du circuit DNS

A présent que l'architecture globale est établie nous passerons à la description des différents blocs la constituant.

Notre système possède trois caractéristiques principales

A. La modularité : Cette caractéristique permet une plus grande souplesse de développement et une facilité de modification du fait que les module sont indépendants les uns des autres; ce qui permet leur la réutilisation.

B. Le contrôle distribué : Le type d'architecture que nous proposons pose des problèmes liés à la communication entre les modules et au contrôle de fonctionnement; pour remédier à ce problème nous avons opté pour un contrôle distribué au lieu d'un contrôle centralisé dans le but de diminuer la complexité et donc de faciliter la gestion

des communications. Comme le montre la figure si dessus, nous avons conçu un module de contrôle qui sera responsable de la gestion des communication au niveau de chaque sous système.

C. L'asynchronisme : Le contrôle distribué a impliqué un fonctionnement asynchrone.

4. 3.1 Bloc Reception

La figure ci après résume et illustre l'algorithme de fonctionnement du bloc *reception*

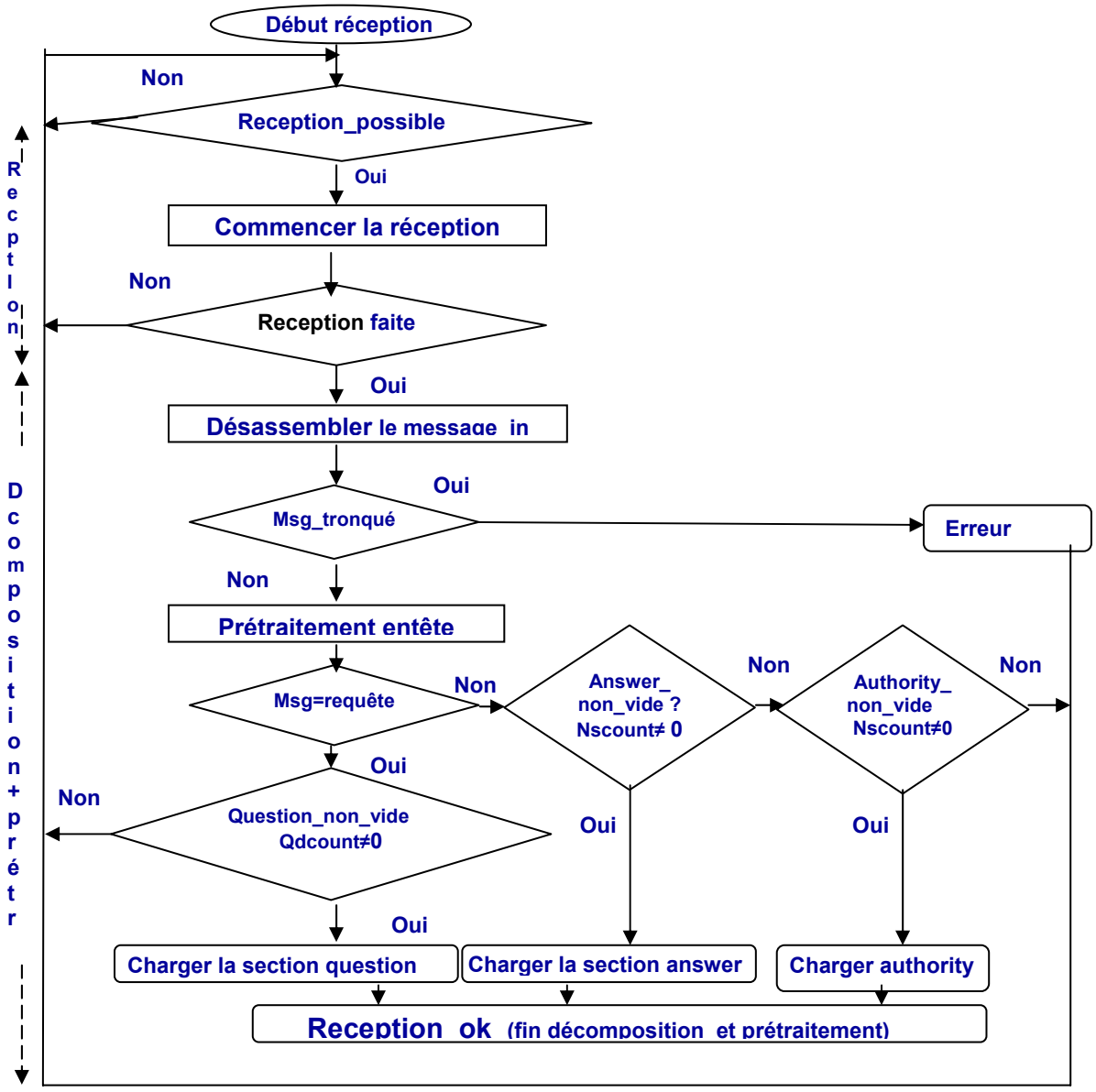


Figure 4.3 : Organigramme réception

A- Description fonctionnelle

La donnée qui constitue généralement la requête est fournie au bloc *reception* via un bus qui est spécifié sur 8 bits.

Sa fonction principale est le stockage de la donnée en entrée et le désassemblage du *message_dns_in* en plusieurs sections et la réalisation de la correspondance entre la chaîne de caractères reçue à travers le signal *message_dns_in* et les différents champs constituant le message DNS. La structure des données véhiculées dans le *message_dns_in* est définie dans le package *serveur_DNS* auquel tous les blocs constituant l'architecture proposée font référence. En effet, dans ce package nous avons décrit l'ensemble des types et fonctions partagées par l'ensemble des modules et particulièrement un type de donnée appelé *message_dns_in* et dont la structure est la même que celle du message DNS présenté dans le chapitre 1.

La fonction décomposition consiste à séparer l'entête de douze octets qui englobe toutes les informations utiles pour la résolution des autre sections du message DNS. L'entête est constituée d'un nombre de champs fixe toujours présents. Particulièrement Le champ *opcode* qui reste l'essentiel de l'information reçue, ce champ renseigne sur le type de requête et donc sur le type de fonction demandée au bloc *resolution*.

Le bloc *reception* est responsable aussi de la transmission de la donnée au bloc *resolution* et fournit les informations nécessaires pour les processus de résolution de noms.

Ce bloc doit localiser toutes ces informations dans la chaîne de caractère reçue en ce référant au types décrit dans le package *serveur_DNS*.

Essentiellement on y trouve les types suivants :

* type de requête (*opcode*)
* nom de domaine pour le quel l'adresse IP est demandée (*qname*) ;
* type et class de l'enregistrement de ressource demandée (*qtype et qclass*).

Comme le montre la figure suivante, le bloc *reception* est composé de sept modules interconnectés

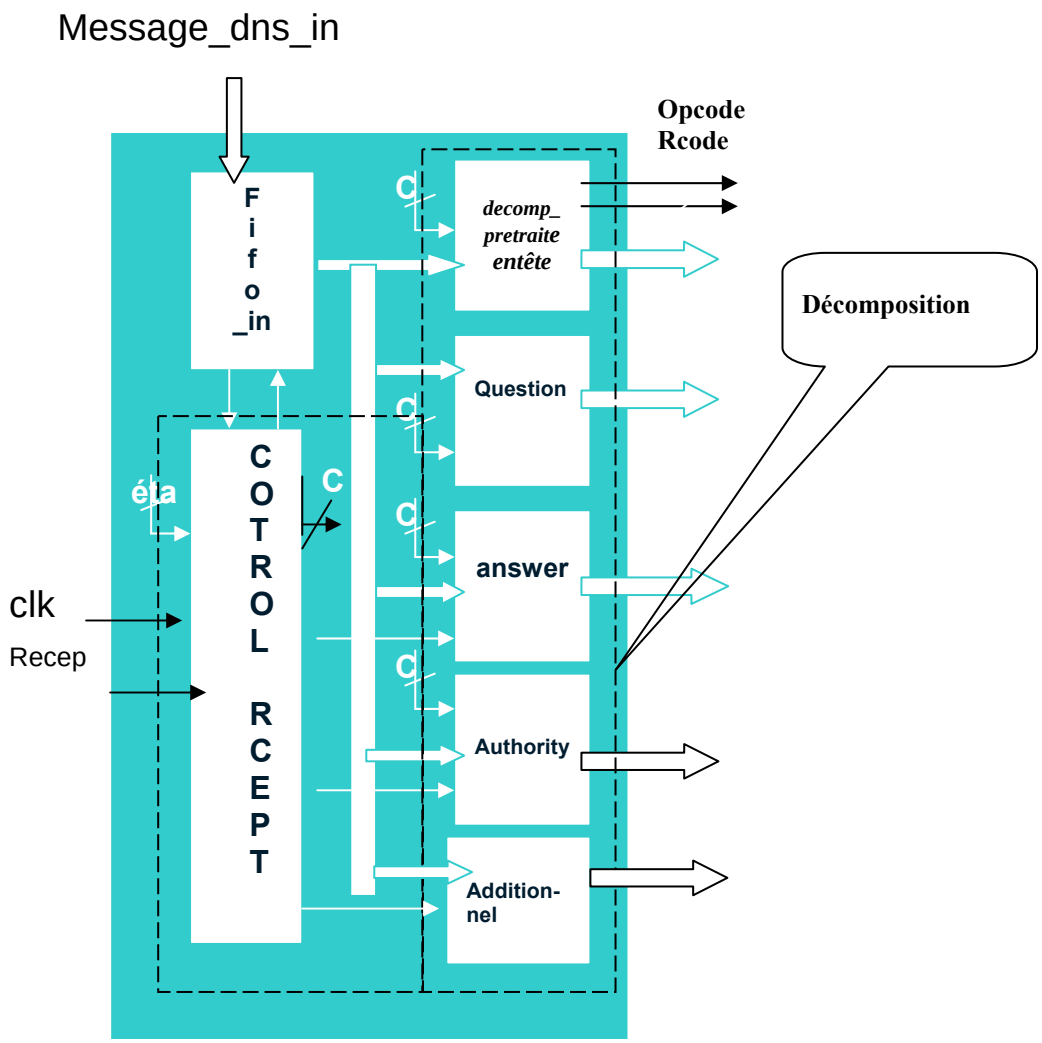


Figure 4.4: Architecture interne du bloc *Reception*

Dans la suite de cette section, nous allons décrire en détail les fonctionnalités de chaque sous module du bloc *reception*. Nous donnerons les résultats de simulation de chaque entité testée séparément.

Remarque : pour tous les tests, la chaîne de caractères est représentée en hexadécimal pour visualiser entièrement la chaîne de caractère (elle est transmise en binaire vers les différents modules).

4.3.1.1 Module fifo_in

4.3.1.1 A Fonctionnement

Fifo_in (*First-In First-Out en entré),* ce qui signifie que le premier mot écrit dans la mémoire sera le premier lu. C'est une fifo circulaire d'une profondeur de 512 mots de 8 bits servant de tampon pour le bloc *reception*.

Le signal *message_dns_in* est transmis via un bus de 8 bits (octet par octet) au module fifo_in. De cette manière, nous stockons des informations jusqu''a ce qu'il ait suffisamment pour que le bloc suivant puisse démarrer.

Ce module inclut quelques signaux internes et communique en fait avec les autres modules du bloc *reception* au moyen de plusieurs signaux de contrôle. Les informations seront stockées, à une place fixe, dans une mémoire constituée d'un tableau de registres. Cette mémoire sera gérée de manière circulaire à l'aide de deux index ; l'un repèrera la position dans laquelle se fera la prochaine écriture alors que l'autre repèrera la position dans laquelle se fera la prochaine lecture. Ces deux index se suivront dans leurs déplacements. Si l'index d'écriture rattrape celui de lecture, alors la fifo sera considérée comme pleine. Dans le cas ou l'index de lecture rattrape celui d'écriture, alors le fifo sera alors vide. Dans la description de la *fifo_in* nous avons utilisé des variables entières pour représenter ces index. La figure 4.5 représente la vue externe de ce module.

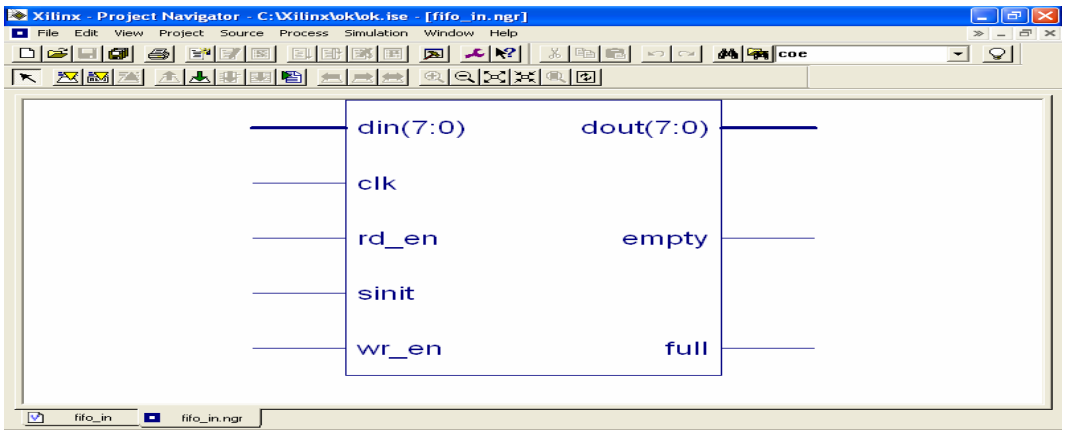


Figure 4.5: Vue externe la *fifo_in*

Nom	Direction	Largeur	Description
Clk	In	1 bit	Horloge.
Sinit	In	1 bit	Remise à zéro, lorsque sinit ='1' la fifo est déclaré vide (cette entrée annule le contenu de la mémoire)
Wr_en	In	1 bit	Entrée active sur front montant, elle correspond à une demande d'écriture.
Rd_en	In	1 bit	Active sur front montant, elle correspond à une demande de lecture.
DIN	In	8 bits	Vecteur de données en entrée, il contient les données qui seront stockées dans la fifo après une demande d'écriture.
Dout	Out	8 bits	Vecteur de données en sortie, il est mis à jour après une demande de lecture.
Empty (vide)	Out	1 bit	Signal d'état est actif (prend la valeur '1') lorsque la fifo est vide.
Full (pleine)	Out	1 bit	Signal d'état est actif (prend la valeur '1') lorsque la fifo est pleine.

Tableau 1 : Entrées et Sorties de la *fifo_in*

La description des entrées et des sorties de la *fifo_in* sont données dans le tableau 1.

4.3.1.1 B Simulation

Pour le test de la *fifo_in*, la chaîne de caractères représentant le *message_dns_in* est envoyé dans la *fifo_in* via un module lecture (lit dans un fichier nommé "mes_dns .dat" les octet successives du *message_dns_in* représenté par une chaîne de caractères de 33 octet).

Le résultat pour ce module de test est donné ci après. Il représente une partie du *message_dns_in.*

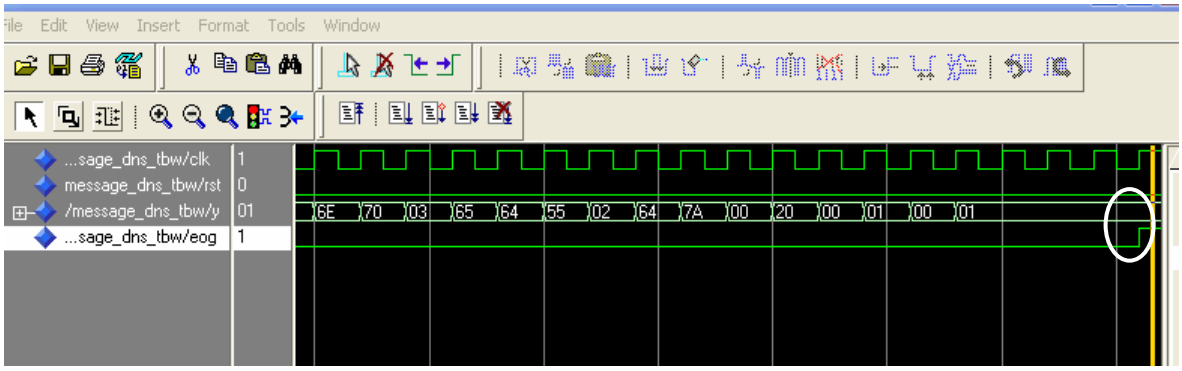


Figure 4.6: Résultat de simulation du module *lecture_fichier*

La sortie *Y* représente le contenu du fichier message_dns qui sera injecté dans le module *fifo_in*. Cette sortie sera l'entrée *din* de la *fifo_in* et le bit '*eog*' passe à '1' à la fin du fichier à lire (la fin du message dns).

Notons que ce module n'est pas inclut dans l'architecture que nous proposons, il n'est utilisé que pour les tests.

On simule l'action du contrôleur en réception en mettant le signal *sinit* à '0', et les le signaux *wr_en* et *rd_en* à '1'.

Les résultats de simulation pour ce bloc sont représentés sur les figures ci après :

1-*fifo_in*, seul (avant l'envoie du message_dns_in)

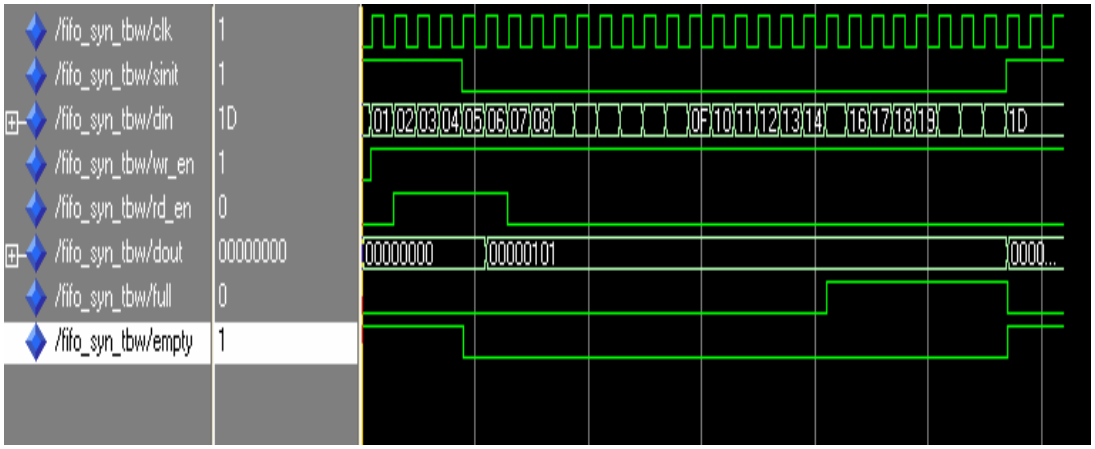


Figure 4.7 : Résultat de simulation du module *fifo_in*

Au moment du front montant de *wr_en* on écrit la donné dans la *fifo_in* et le bit *empty* passe à '0' (la fifo n'est plus vide). Dès que *rd_en* passe à '1', on remarque qu'on a des données en sortie et *full* passe à '1' lorsque la *fifo_in* est pleine puisque

nous avons mis le bit *rd_en* à '0', le signal d'état *full* passe à '0' quand *rd_en* sera égale à '1'.

2-Les figures suivantes représentent la *fifo_in* avec le *message_dns_in* en entrée.

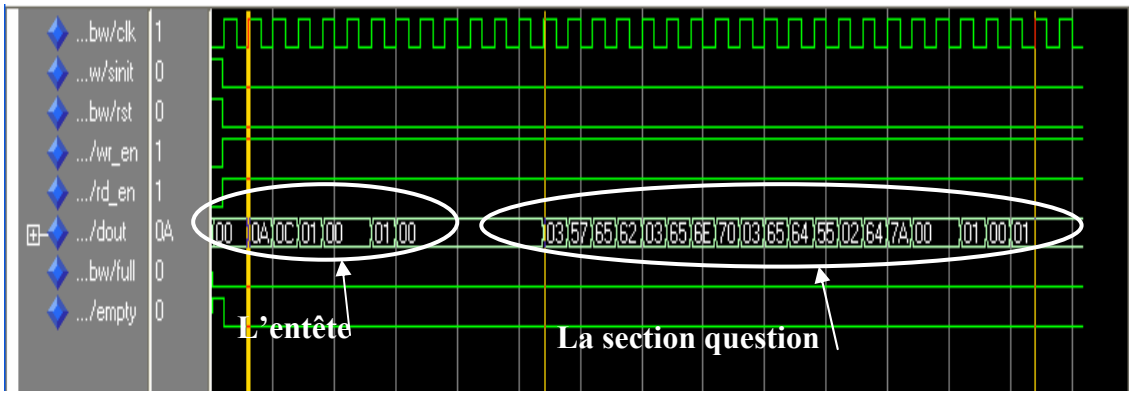


Figure 4.8.a: Résultat de simulation du module *fifo_in* (message_dns en entrée)

Représentation en ASCII pour mieux voir le champ qname

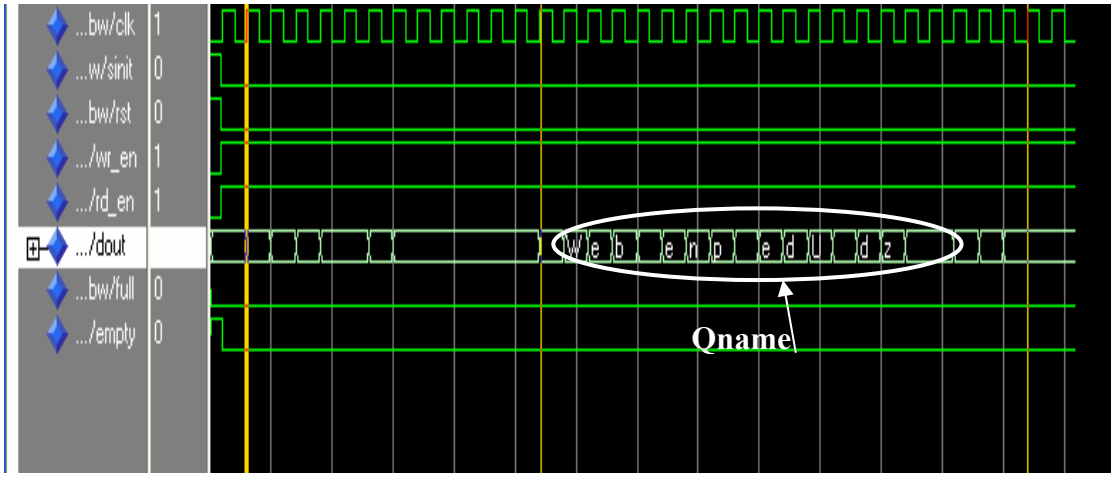


Figure 4.8.b:Résultat de simulation du module *fifo_in* (représentation ascii)

4.3.1.2 Sous bloc décomposition

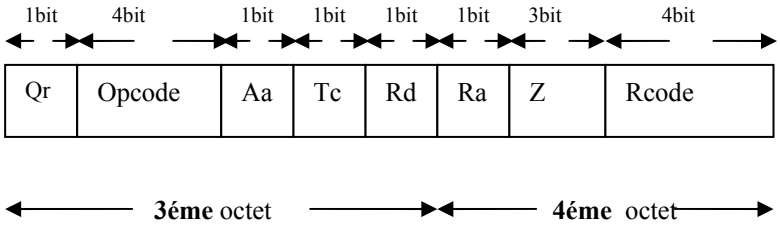
4.3.1.2.1 Module decomp_pretraitement_ entete

4.3.1.2.1 A Fonctionnement

Ce module est un circuit de décomposition, il inclut une mémoire de douze mots de 8 bits accessibles en lecture et en écriture. Le contrôle de ce module est effectué par les signaux suivants : une commande d'écriture et une commande de lecture. Ces commandes sont gérés par le module *controle_reception* (qui sera décrit ultérieurement) et sont synchronisés par une horloge « *clk* ». En plus des signaux de lecture et d'écriture, nous avons défini aussi le signal *decomp_entete* qui correspond à un ordre de décomposition. Le processus de décomposition représente la spécificité de ce module.

La décomposition de l'entête consiste à faire la correspondance entre les 12 octet enregistrés après une opération d'écriture dans un tableau de 12 mots de 8bits, une variable « ent » est déclaré comme variable de type tableau , modélisant le plan de l'entête et les différents champs constituant l'entête du message DNS. La structure des données est définie dans le package *serveur_DNS* précisément dans le record *entête* où nous avons spécifié le type et la taille de chaque champ de l'entête DNS. L'information que véhicule l'entête sera par la suite traitée par le bloc *resolution*.

Le rôle de ce module est de décomposer le troisième et le quatrième octet en plusieurs champs de longueurs différentes et assigner chaque champ à une partie du mot correspondant dans le tableau entête dont le champ le plus significatif « *opcode* » (renseigne sur le type de requête). Les champs nécessaires pour le processus de résolution et représentant les entrées du bloc *resolution* sont localisés dans la tableau «entête » interne et affecté au ports correspondants en sortie. Les champs ayant une largeur supérieure à 8 bits tels que le champ ID de 16 bits (une concaténation du premier et du deuxième octet) seront obtenus par concaténation. De la même manière sont définies les champs *qdcount, anqount, nscount et arcount*. Notons que par rapport aux valeurs de ces champs se fera la lecture de la suite du message.
L'association des différentes valeurs contenues dans le 3ème et le 4ème octet aux champs de l'entête se fera suivant la décomposition représentée ci-dessous tandis que l'association des autres champs se fera par concaténation d'octets deux à deux.



La figure 4.9 obtenu avec l'outil RTL shematic représente la vue externe de ce module.

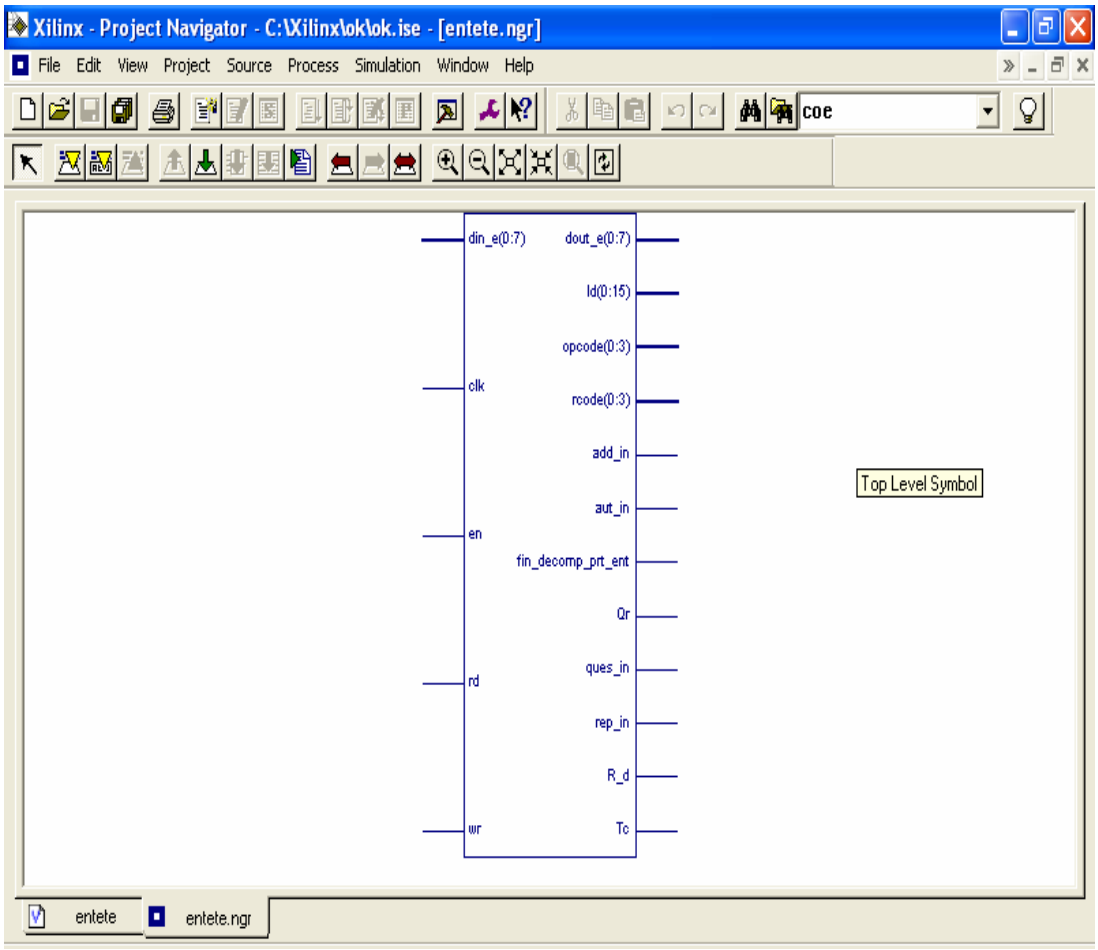


Figure 4.9 : Vue externe du module *decomp_pretraitement_ entête*

Le Tableau 2 récapitule la description des signaux d'entrées/sorties du module *decomp_pretraitement_ entête.*

Remarque : *les sortie qr, tc, r_d, opcode, rcode, id correspondent à une extraction des Champs correspondants dans l'entête du message DNS reçus.*

Nom	Direction	Largeur	Description
Clk	In	1 bit	Horloge active haute .
En=Decmp_in	In	1 bit	Entrée de sélection active sur front montant correspondant aussi à un ordre de décomposition
Wr	In	1 bit	Entrée active sur front montant, elle correspond à une demande d'écriture.
Rd	In	1 bit	Active sur front montant, elle correspond à une demande de lecture.
Din_e	In	8 bits	Ce vecteur est la donnée à insérer dans ce module après une demande d'écriture
ID	Out	16 bits	Vecteur de 16 bits, il correspond à une concaténation du premier et du deuxième octet, et représente le champ id du message DNS reçue.
Dout_e	Out	8 bits	Vecteur de données en sortie, représente la sortie retirée de ce module
Qr	Out	1 bit	Sortie permettant d'indiquer s'il s'agit d'une requête ou d'une réponse
Tc	Out	1 bit	Sortie qui prend la valeurs '1' quant le message est tronqué (correspond à une extraction du bit Tc correspondant dans l'entête du message reçus)
R_d	Out	1 bit	Sortie qui prend la valeurs '1' quant la récursivité est demandée
Opcode	Out	4 bits	Sa valeur spécifie le type de requête
Rcode	Out	4 bits	Sa valeur spécifie le type de réponse
Ques_in	Out	1 bit	Sortie qui prend la valeurs '1' quant la section « question » n'est pas vide (qdcount /= 0)
Rep_in	Out	1 bit	Sortie qui prend la valeurs '1' quant la section « answer » n'est pas vide (ancount /= 0)
Aut_in	Out	1 bit	Sortie qui prend la valeurs '1' quant la section « authority » n'est pas vide (nscount /= 0)
Add_in	Out	1 bit	Sortie qui prend la valeurs '1' quant la section« additionnal» n'est pas vide (arcount /= 0)

Tableau 2 : Entrées et Sorties du module *decomp_pretraitement_ entête*

4.3.1.2.1 B Simulation

Le résultat pour ce module est donné si après. Il représente la partie entête du *message_dns_in*. Comme on peut le remarquer sur le troisième octet (8A en hexa) sont extraits les champs *qr (1 bit)*, et *opcode* de 4 bits. Les champs *rcode* (4 bits) et les bits *Tc, rd* sont extraient de l'octet suivant (80 en hexa). Notons qu'on ne localisera pas le reste des champs dans ses deux octets car ils sont inutiles pour le bloc *resolution*.

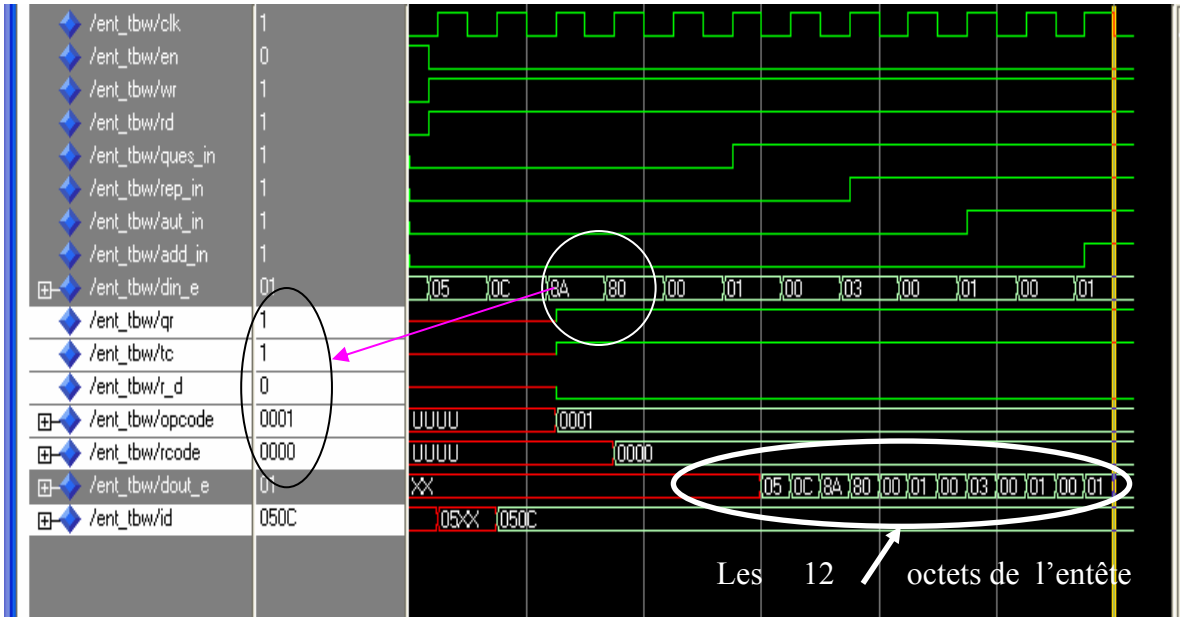


Figure 4.10.a : Résultat de simulation du module *decomp_pretraitement_ entete*

La fenêtre 'structure' si après permet la vérification de la correspondance entre les données contenues dans la chaîne d'octets et les éléments de l'entête DNS. On remarque que les 12 octets en entrée sont décomposés en tranches de largeurs différentes et sont assignées aux champs correspondants dans le record entête.

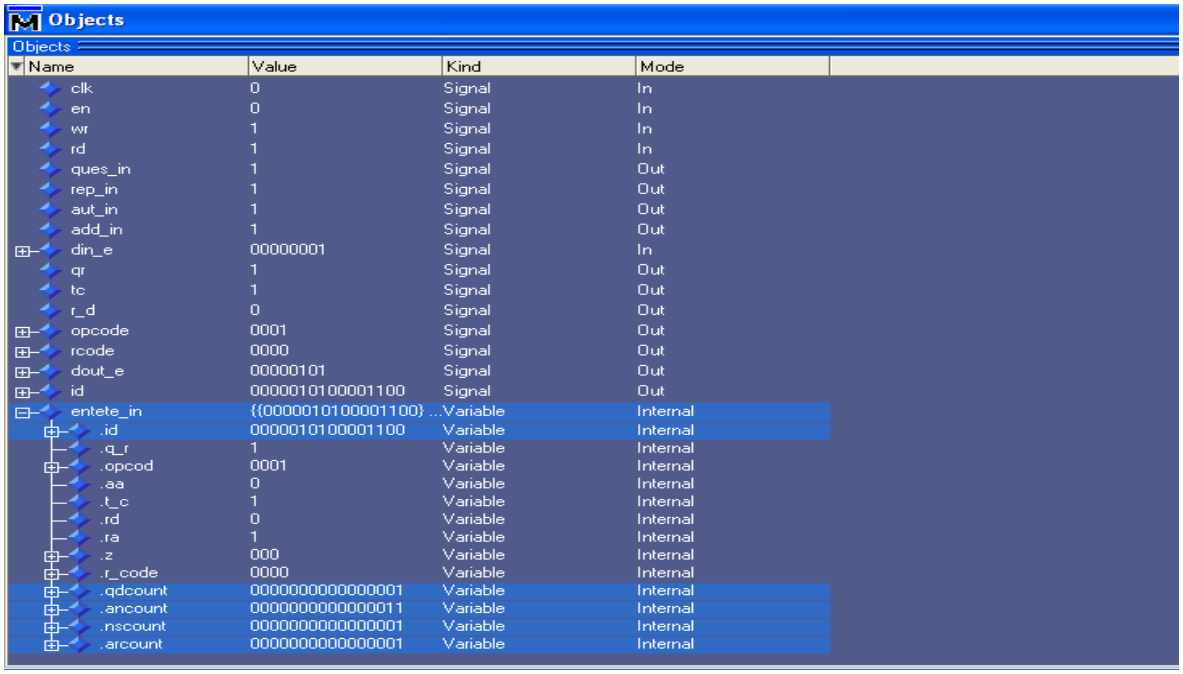


Figure 4.10.b : Résultat de simulation du module decomp_pretraitement_ entête

4.3.1.2.2 Module question

4.3.1.2.2 A Fonctionnement

Ce module représente la section question. La donnée qu'il mémorise est de taille variable spécifiée en mots de 8 bits. Elle est structurée en zones de façon à émuler la structure de la section question du message DNS. Le plan de cette mémoire est un tableau structuré en zones; la première zone stockera le champ QNAME. Ce dernier spécifie le nom d'hôte ou de domaine sur lequel porte la question. La longueur de ce champ est variable allant de 1 octet à 254 octets au maximum. Son format est représenté si dessous

Nombre d'octet	1er identifiant	Nb_octet	2eme identif	----------	Nb_octet	Neme identif

Format d'un nom dans le module question

Les deux autres zones sont Qtype (type du RR recherché) et Qclass (la class du RR recherché). Ils sont représentés par deux octets chacun et sont obtenus par une opération de concaténation.

Qclass est la concaténation des deux derniers octets du tableau question

Qtype est la concaténation des deux octets au dessus. Nous avons choisi une organisation interne sous forme de zones afin de séparer les trois informations que porte la section question. La fonction principale de ce module est l'extraction du champ question qui sera l'entrée du bloc type *resolution*. La vue externe de ce module est comme suit :

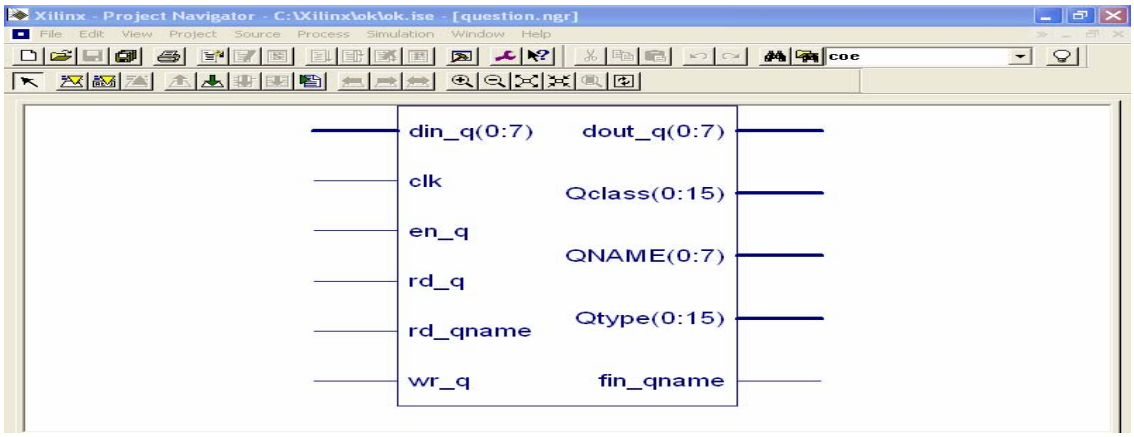


Figure 4.11:Vue externe du module *question*

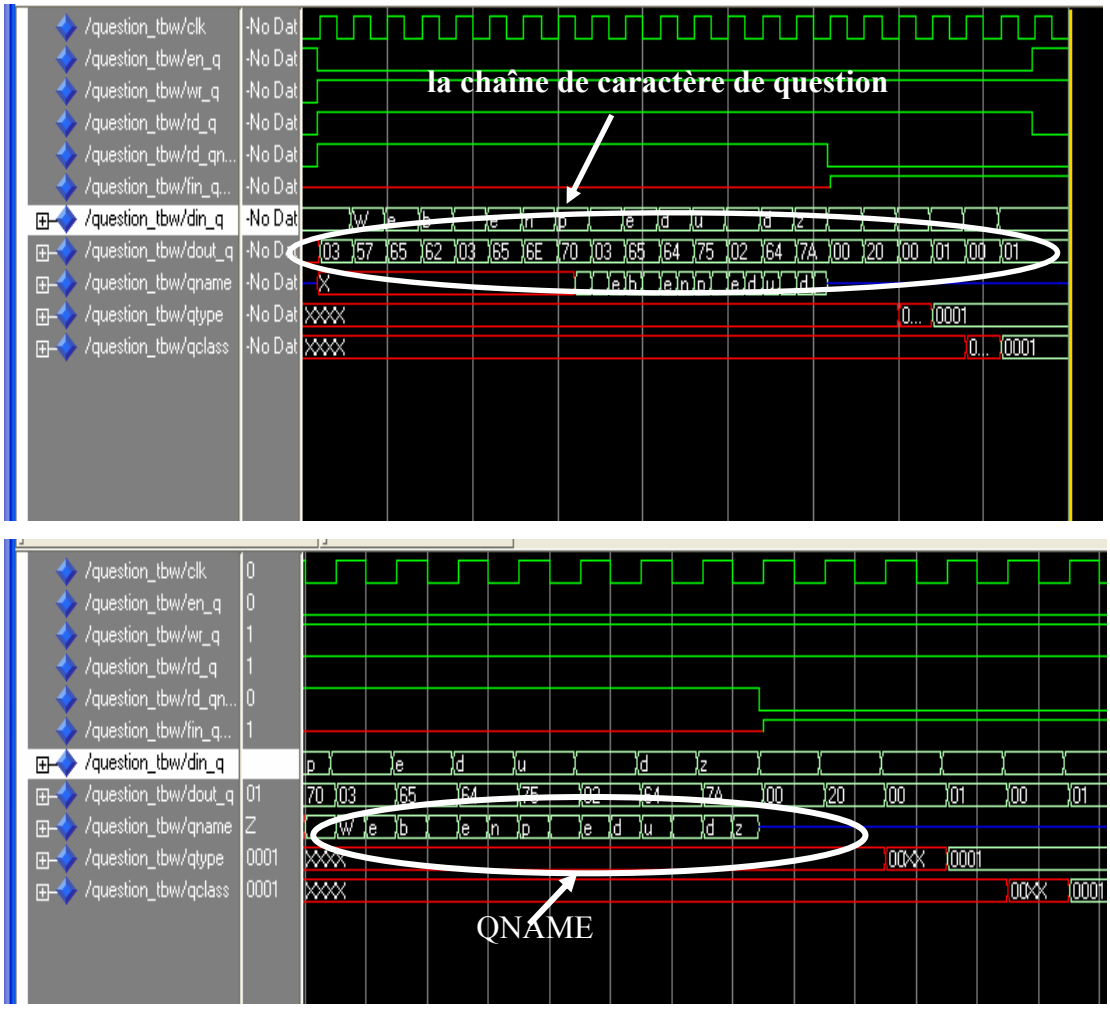
Nom	Direction	Largeur	Description
Clk	In	1 bit	Horloge active haute.
En_q	In	1 bit	Entrée de sélection active sur front montant
Wr_q	In	1 bit	Entrée active sur front montant, elle correspond à une demande d'écriture.
Rd_q	In	1 bit	Active sur front montant, elle correspond à une demande de lecture.
Din_q	In	8 bits	Ce vecteur est la donnée à insérer dans question après une demande d'écriture
Qtype	Out	16 bits	Vecteur de 16 bits, il correspond à une concaténation des deux octets, et représente le champ qtype du message DNS reçue.
Qclass	Out	16 bit	Vecteur de 16 bits, il correspond à une concaténation des deux derniers octets, et représente le champ qclass du message DNS reçue.
Fin_q	Out	1 bit	Sortie qui prend la valeur '1' à la fin qname

Tableau 3 : Entrées et Sorties du module question

Le Tableau 3 récapitule la description des signaux d'entrée/sortie du module *question*

4.3.1.2.2 B Simulation

Pour le test de l'entité *question*, nous avons spécifié la longueur du champ *qname* à 16 octets. *Qname* sur lequel porte la question est « web.enp.edu.dz », on peut remarquer sur les figures ci-dessous l'extraction de ce champ de la chaîne de caractères.



Figures 4.12.a : Résultat de simulation du module question

On voit bien en sortie le champ *qname* de 16 octets. Notons aussi que pour ne pas extraire de la chaîne question que le champ qname, le bus sera de nature « trois états » (donc en haute impédance à la fin de *qname*)

Cette structure est bien illustrée dans la fenêtre structure de Modelsim représentée par la figure suivante.

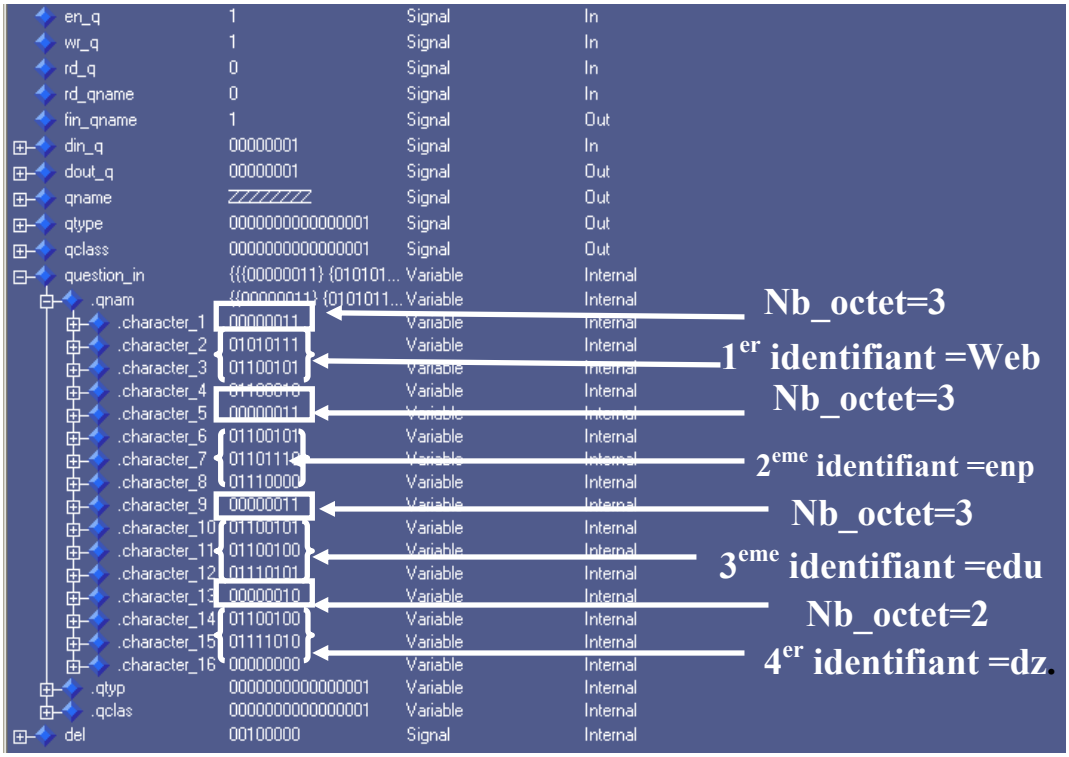


Figure 4.12.b : Résultat de la simulation du module *question* (fenêtre structure)

4.3.1.2.3 Module answer

4.3.1.2.3 A Fonctionnement

Ce module est une entité de mémorisation qui sert à stocker la donnée de la section answer du message DNS.

L'organisation interne de ce module est semblable à celle du module *question*. La chaîne d'octets stockée dans cette mémoire représente les RRs dont la structure est définie dans le package *serveur_DNS*. Chaque zone de cette mémoire est assignée à son champ correspondant dans le RR reçus.

Ce module a donc pour rôle de préciser la donnée (véhiculées par une partie de la chaîne de caractères) à associer à chaque élément du RR.

La vue externe du module *answer* est comme suit :

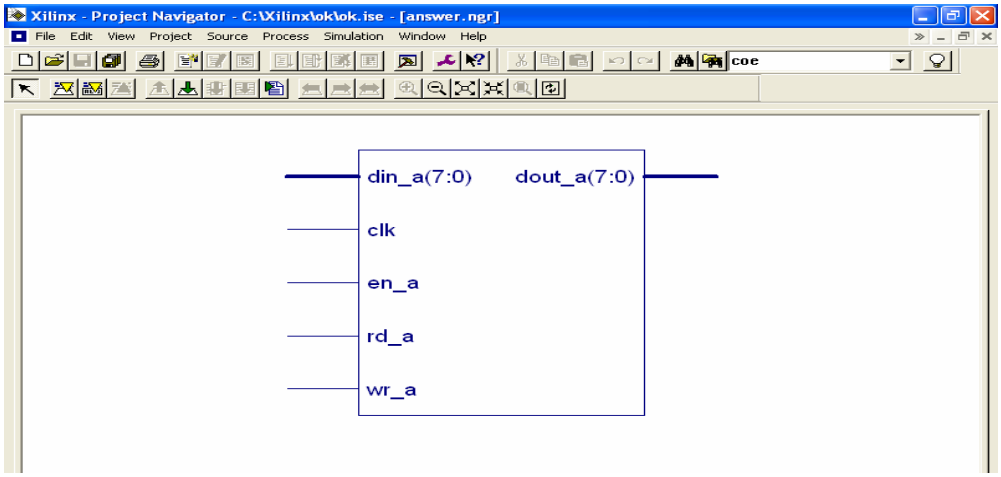


Figure 4.13 : Vue externe du module *answer*

De même, le tableau 4 récapitule la description des signaux d'entrée/sortie du module *answer*

Nom	Direction	Largeur	Description
Clk	In	1 bit	Horloge active haute du système.
En_a	In	1 bit	Entrée de sélection active sur front montant
Wr_a	In	1 bit	Entrée active sur front montant, elle correspond à une demande d'écriture.
Rd_a	In	1 bit	Active sur front montant, elle correspond à une demande de lecture.
Din_a	In	8 bits	Ce vecteur est la donnée à insérer dans *answer* après une demande d'écriture
Dout_a	Out	8 bits	Vecteur de données en sortie, représente la sortie retirée de answer

Tableau 4 : Entrées et Sorties du module *answer*

Nous ne ferons pas de description pour les modules *authority* et *additionnal* car ils sont identiques dans leur organisation interne au module *answer*. Aussi ces trois modules auront la même vue externe.

4.3.1.2.3 B Simulation

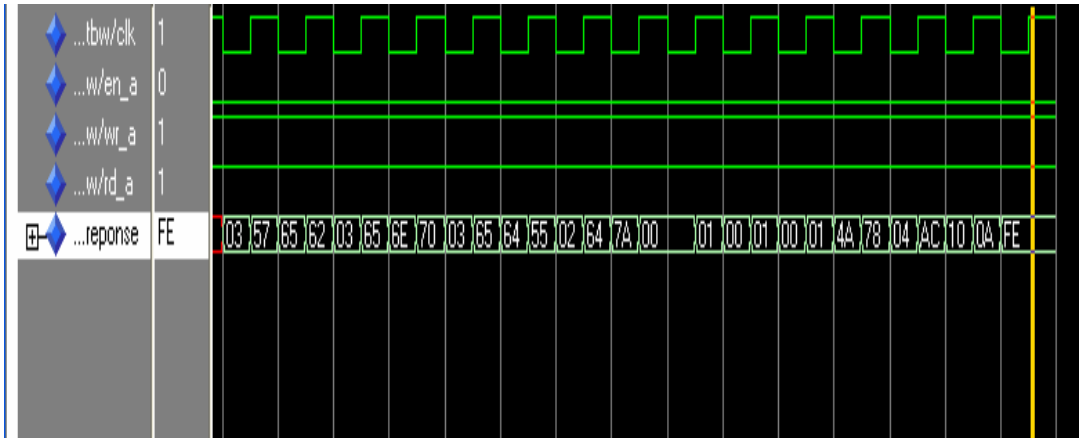


Figure 4.14: Résultat de la simulation du module *answer*

4.3.1. 3 Module control_reception

4.3.1. 3 A Fonctionnement

Ce contrôleur renferme une machine à états illustrée par la Figure 4.15. Il faut préciser que la machine à états devra être de type *MEALY*. Quelque soit, l'état dans lequel le contrôleur se trouve, il doit interrompre tout traitement dès qu'il voit que le signal « *recep*» est égal à '0' et se mettre en état « idle » s'il ne l'est pas déjà. Le bit « *recep* » est une entrée indiquant qu'il y a réception d'un message, le signal « *recep* » devra être à '1' durant toute la durée de la réception.

Une transition d'un état à un autre est caractérisée par une condition et par l'affection de valeurs aux signaux de sortie. La figure ci dessous illustre les conditions associées à chaque transition de la machine à états ainsi que les valeurs affectées aux signaux de sorties du contrôleur.

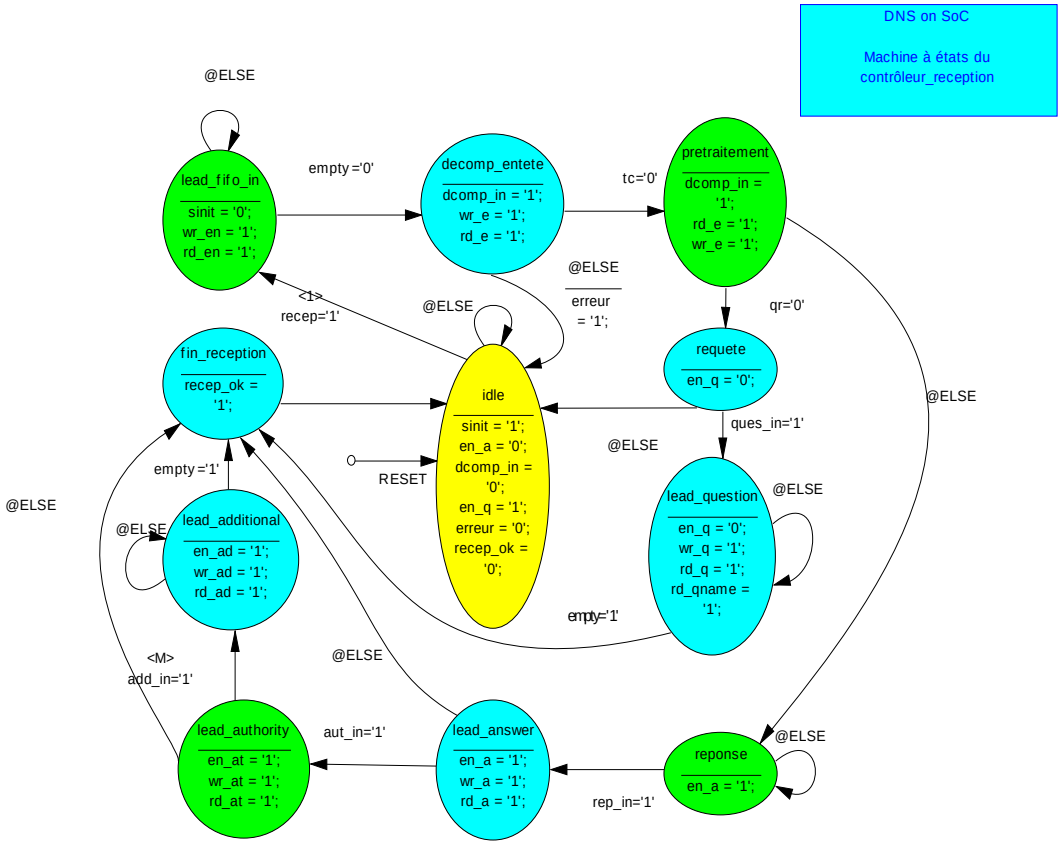


Figure 4.15: Machine à état du module *control_reception*

La vue externe du module *control_reception* est représentée ci après

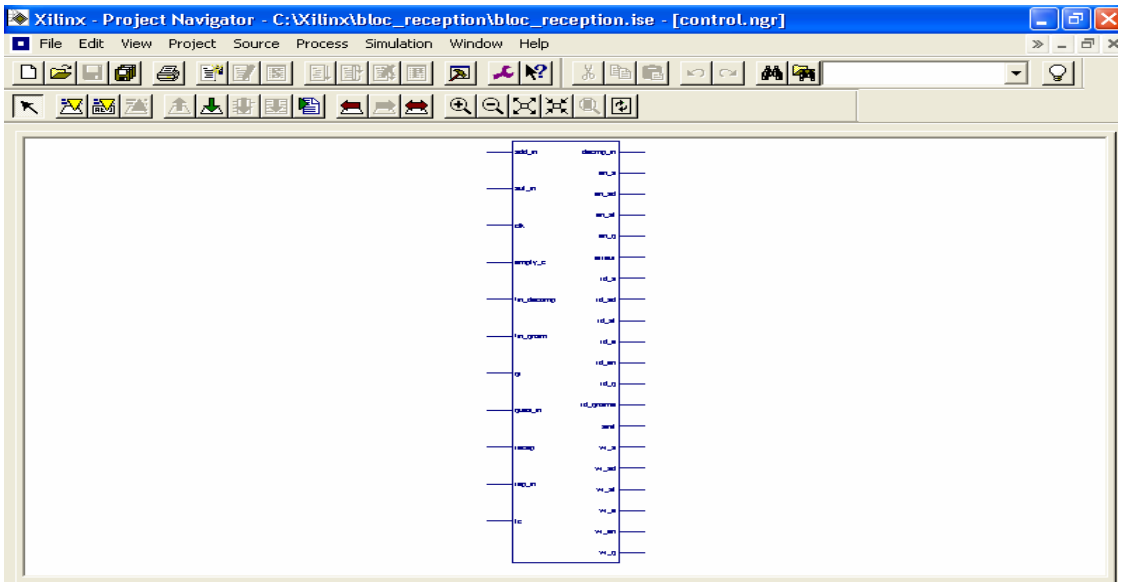


Figure 4.16 : Vue externe du module *control_reception*

4.3.1.3 *B Simulation*

Voici les résultats de simulation pour ce module :

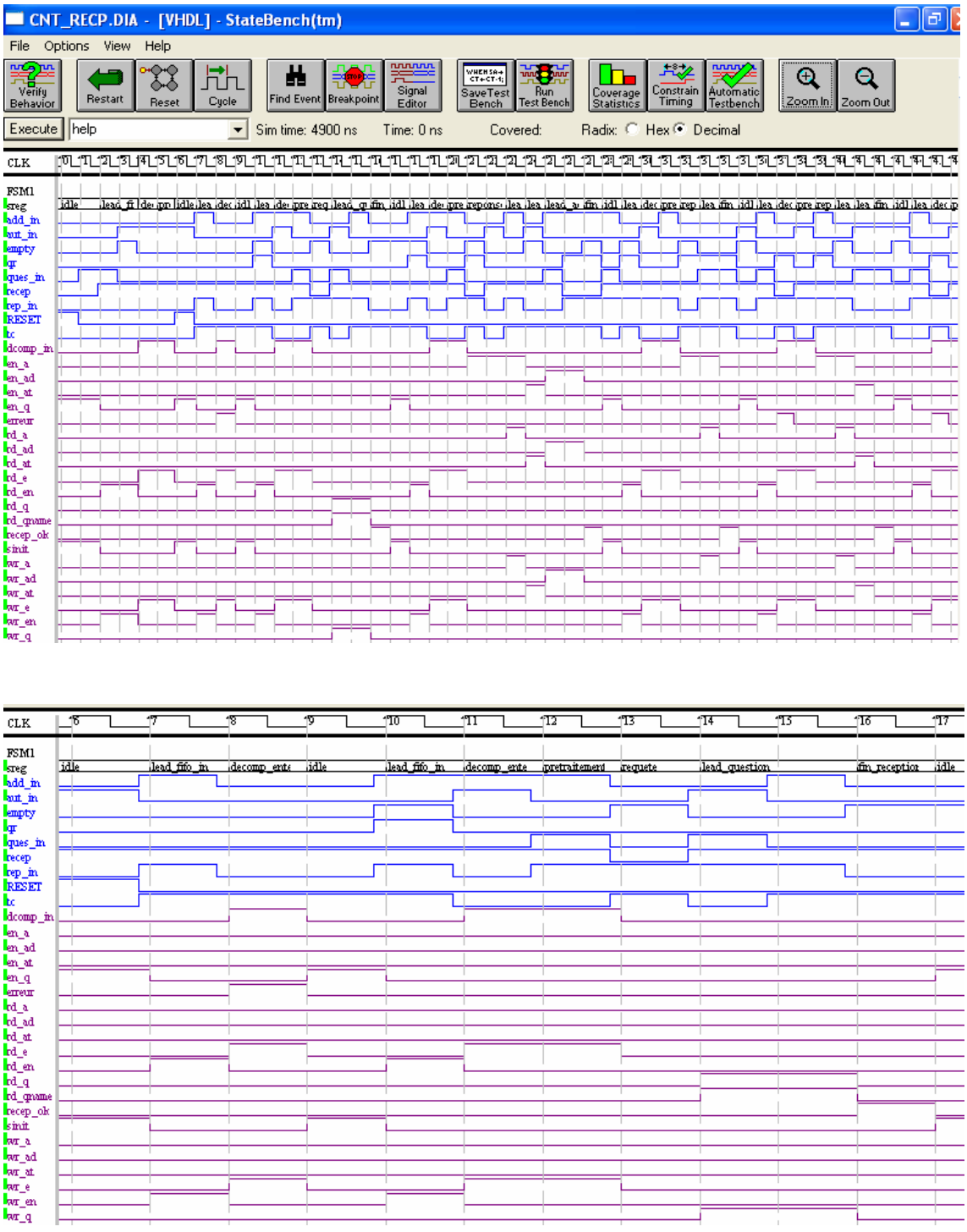


Figure 4.17 : Résultats de simulation du module *control_reception*

L'interconnexion des modules composant le sous système réception, par une description structurelle à conduit au schéma de la figure 4.18.

Vu externe du bloc *reception* généré avec l'outil RTL schematic

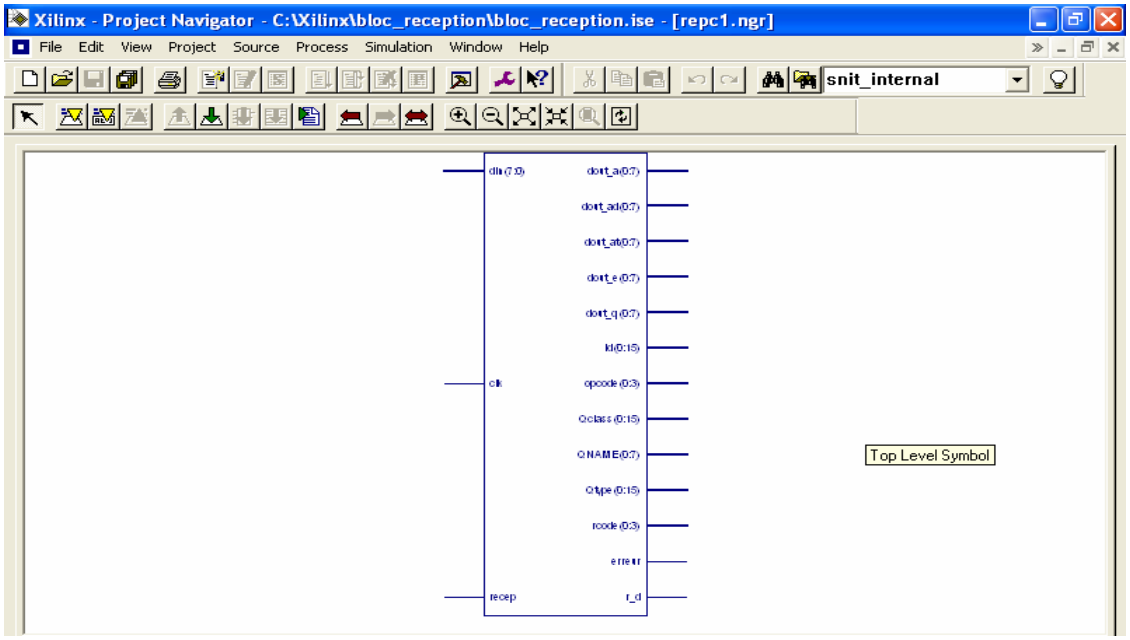


Figure 4.18 : Circuit *reception* synthétisé

La structure interne du sous système réception est donnée par la figure suivante

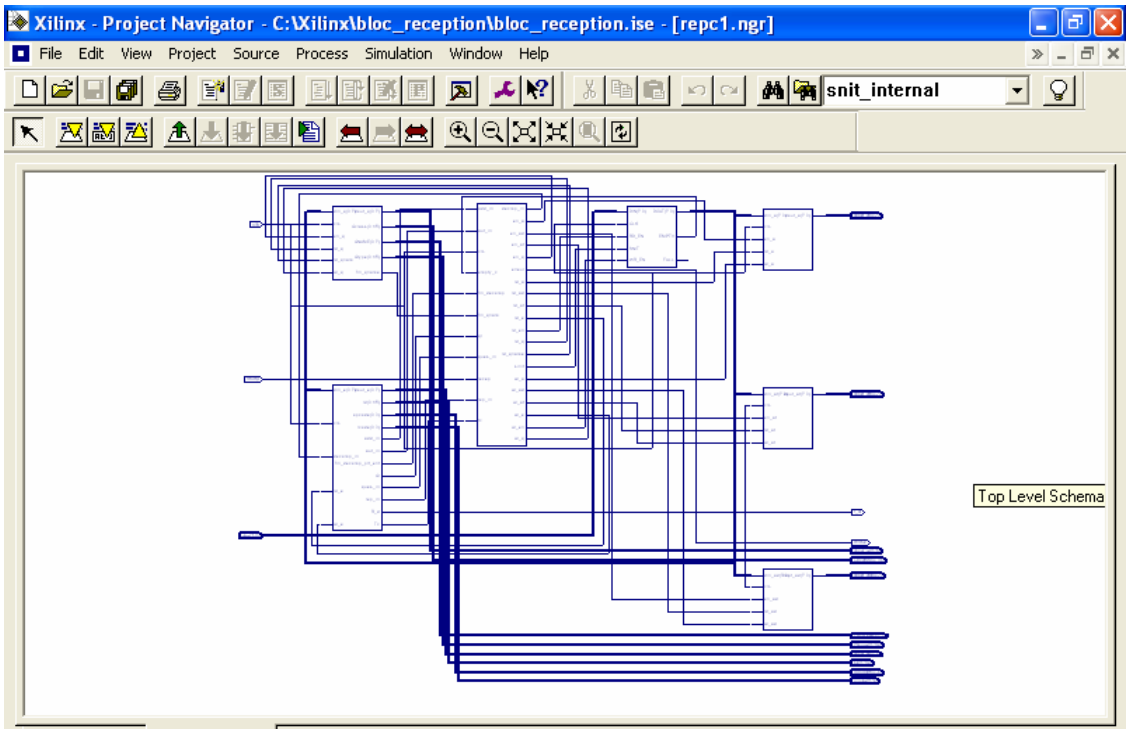


Figure 4.19 : Structure interne du circuit (bloc *reception*) synthétisé

4. 3.2 Bloc Configuration

4. 3.2 Fonctionnement

La figure si après représente la structure interne du bloc *configuration*

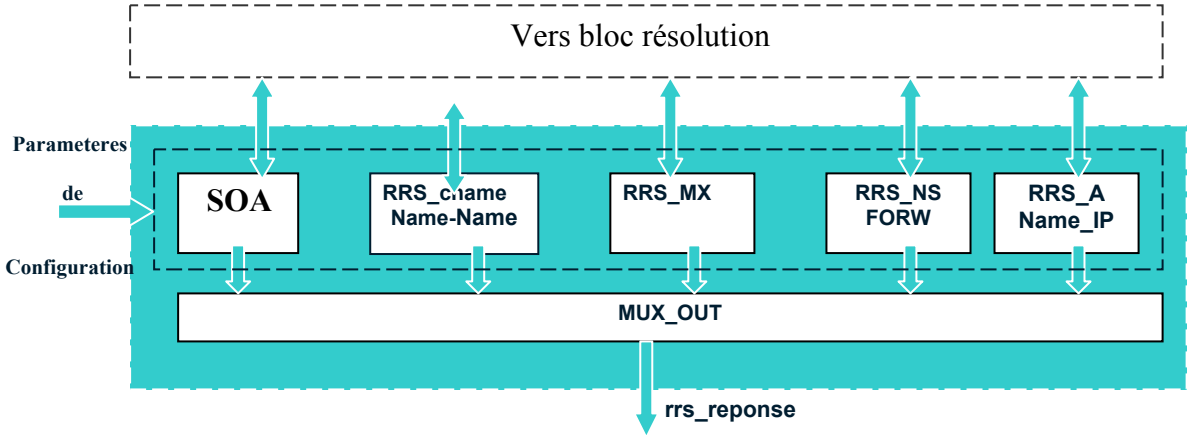


Figure 4.20: Structure interne du bloc *configuration*

Le bloc *configuration* a pour rôle de recevoir les informations concernant la configuration du serveur DNS comme les paramètres réseau par exemple les adresses IP des différents serveurs esclaves s'il existe.

Il définit aussi tous les autres paramètres "techniques et administratifs" des zones pour lesquelles le serveur est autoritaire, ces paramètres sont:

- Le nom du DNS primaire (nom du serveur faisant autorité pour la zone dans notre cas enp.edu.dz)
- Adresse mail : email du responsable technique de la zone en remplaçant le "@" de l'email par un "**.**".
- L'adresse IP du serveur maître.
- Les données techniques des la zone [3]: serial Retry, refrech, expire, TTL.

La structure de chaque information que le bloc *configuration* peut recevoir est définie dans le package *seveur_DNS* .En effet nous avons déclaré dans ce package le type

record appelé « SOA » sa structure est similaire à celle du début d'une autorité des serveur DNS software .

Le nom de domaine dans la structure que nous avons décrit est non pas un type « string »mais une zone mémoire de largeur variable ou chaque octet représente un caractère du nom de domaine, les autres paramètres sont des vecteurs de longueurs variables. Cette structure est bien illustrée dans l'exemple que nous avons choisi pour la simulation. Il s'agit de la configuration d'un serveur DNS pour « enp.edu.dz ».Aussi ce bloc, devra assigner ces informations (fournie dans une chaîne de caractères) aux champs correspondant dans le SOA et de préciser la donnée à associer à chaque sortie de ce module pour les envoyer en suite, au bloc *controle_resolution* qui joue le rôle d'analyseur des données transférées au bloc *resolution* dans notre système.

Les RRs stockés dans le sous bloc données_rrs sont reparties entre les modules selon le type de donnée quelles renferme.

Les modules constituant le sous bloc données_rrs sont :

RRs_cname (stocke les enregistrements de type cname « Name_name »), **RRs_MX** (stocke les enregistrements de type MX « Mail Exchange»), **RRs_NS** (stocke les enregistrements de type NS nom du serveur « name serveur »), **RRs_A** (stocke les enregistrements de type A « NAME_IP »).

Le module *MUX_OUT* : le rôle de ce module est de sélectionner le port de sortie associé à chaque module de données_RRs selon la valeur *out_sel* qui sera envoyée au multiplexeur par le module *control_resolution* (qui sera décrit ultérieurement) après la fin de la fonction de résolution.

La sortie du multiplexeur véhiculant le RR de réponse correspondant à la question, sera l'entrée du module *answer* du bloc *emission*

4. 3.2.1 Sous bloc données_RRs

1-Module SOA

Vue externe du module *SOA*

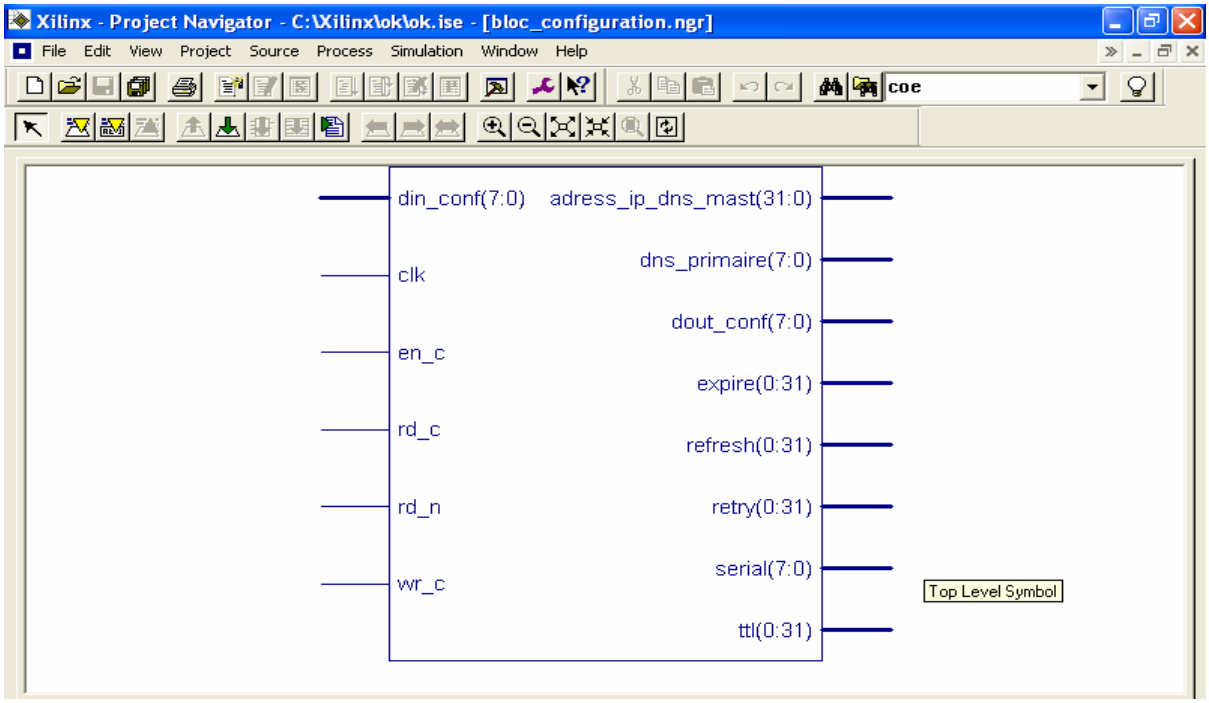
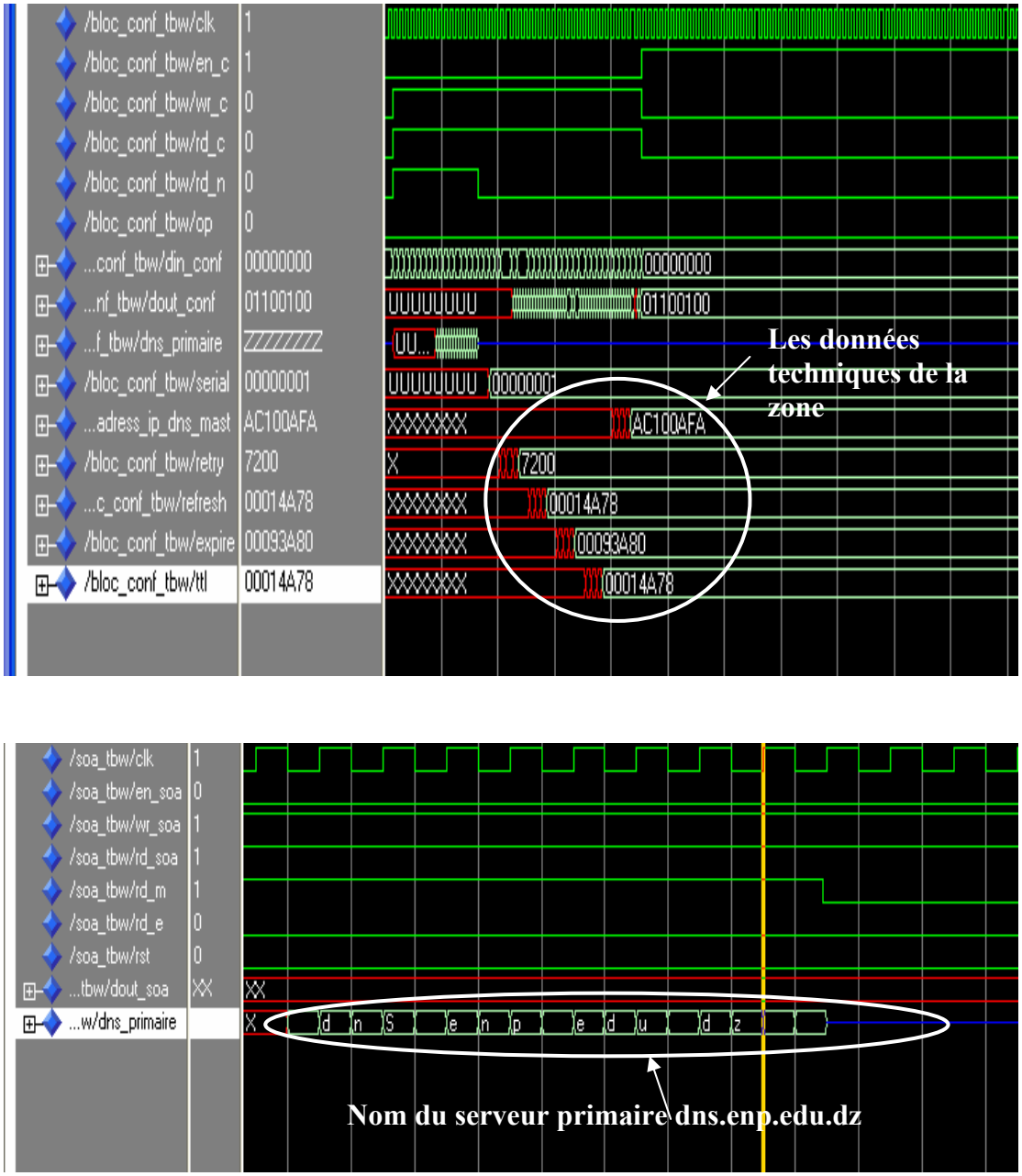


Figure 4.21 : Vue externe du module *SOA*

4. 3.2.2 Simulation

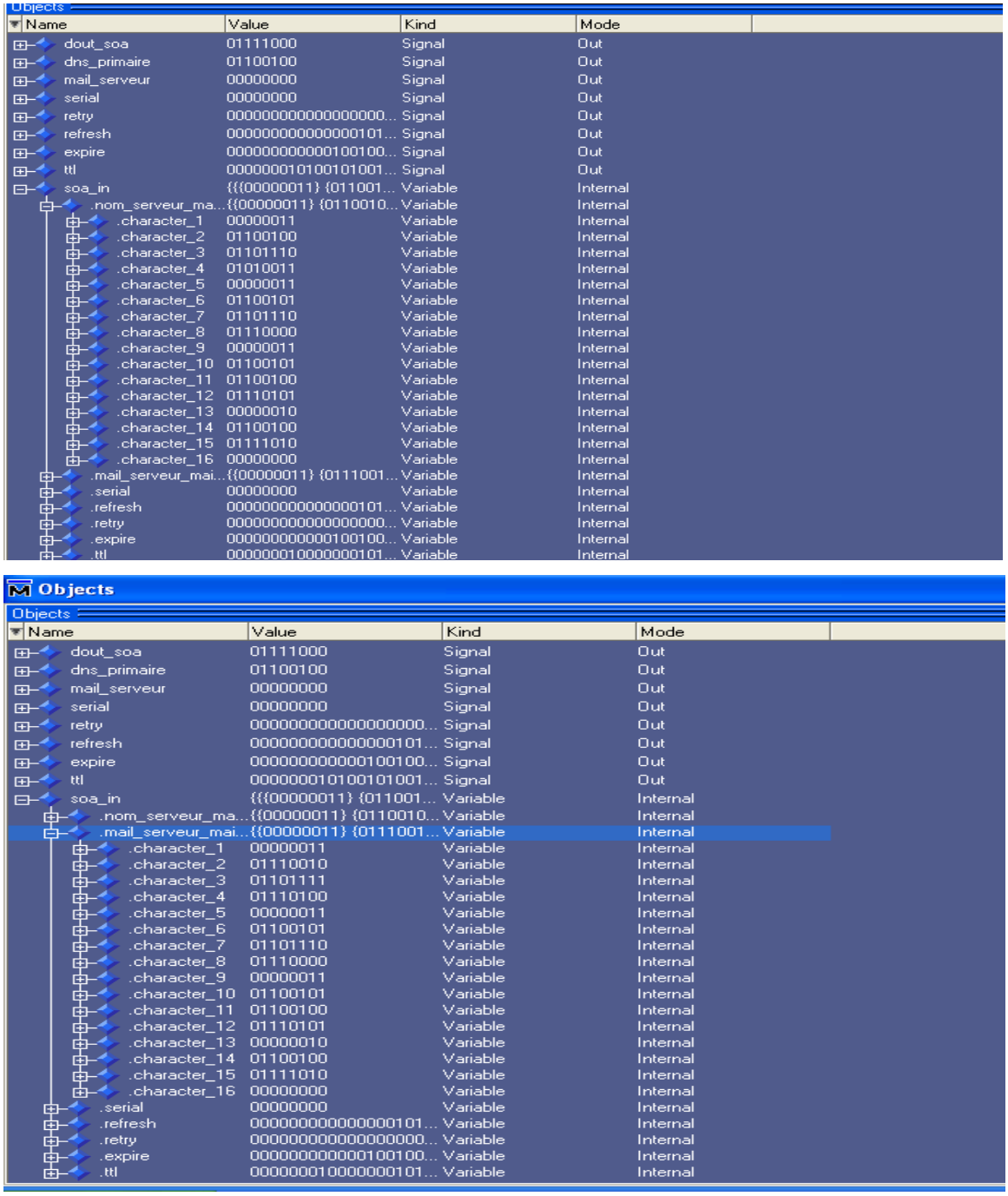
On voit que les valeurs de données correspondantes aux champs du SOA sont bien assignées aux sorties correspondantes de ce module et particulièrement le champ dns_primaire qu'on peut voire sur l'autre figure.

On peut dire que l'association des données aux paramètres de configuration correspondants est vérifiée.



Figures 4.22.a: Résultat de simulation du module *SOA*

La structure représentant l'organisation interne du module *SOA* est bien illustrée dans la fenêtre structure de Modelsim représentée par la figure suivante.



Name	Value	Kind	Mode
dout_soa	01111000	Signal	Out
dns_primaire	01100100	Signal	Out
mail_serveur	00000000	Signal	Out
serial	00000000	Signal	Out
retry	00000000000000000000...	Signal	Out
refresh	00000000000000000101...	Signal	Out
expire	00000000000000100100...	Signal	Out
ttl	00000001010010010001...	Signal	Out
soa_in	{{{00000011} {011001...	Variable	Internal
.nom_serveur_ma...	{{00000011} {0110010...	Variable	Internal
.character_1	00000011	Variable	Internal
.character_2	01100100	Variable	Internal
.character_3	01101110	Variable	Internal
.character_4	01010011	Variable	Internal
.character_5	00000011	Variable	Internal
.character_6	01100101	Variable	Internal
.character_7	01101110	Variable	Internal
.character_8	01110000	Variable	Internal
.character_9	00000011	Variable	Internal
.character_10	01100101	Variable	Internal
.character_11	01100100	Variable	Internal
.character_12	01110101	Variable	Internal
.character_13	00000010	Variable	Internal
.character_14	01100100	Variable	Internal
.character_15	01111010	Variable	Internal
.character_16	00000000	Variable	Internal
.mail_serveur_mai...	{{00000011} {0111001...	Variable	Internal
.serial	00000000	Variable	Internal
.refresh	00000000000000000101...	Variable	Internal
.retry	00000000000000000000...	Variable	Internal
.expire	00000000000000100100...	Variable	Internal
.ttl	00000001000000000101...	Variable	Internal

Objects

Name	Value	Kind	Mode
dout_soa	01111000	Signal	Out
dns_primaire	01100100	Signal	Out
mail_serveur	00000000	Signal	Out
serial	00000000	Signal	Out
retry	00000000000000000000...	Signal	Out
refresh	00000000000000000101...	Signal	Out
expire	00000000000000100100...	Signal	Out
ttl	00000001010010010001...	Signal	Out
soa_in	{{{00000011} {011001...	Variable	Internal
.nom_serveur_ma...	{{00000011} {0110010...	Variable	Internal
.mail_serveur_mai...	{{00000011} {0111001...	Variable	Internal
.character_1	00000011	Variable	Internal
.character_2	01110010	Variable	Internal
.character_3	01101111	Variable	Internal
.character_4	01110100	Variable	Internal
.character_5	00000011	Variable	Internal
.character_6	01100101	Variable	Internal
.character_7	01101110	Variable	Internal
.character_8	01110000	Variable	Internal
.character_9	00000011	Variable	Internal
.character_10	01100101	Variable	Internal
.character_11	01100100	Variable	Internal
.character_12	01110101	Variable	Internal
.character_13	00000010	Variable	Internal
.character_14	01100100	Variable	Internal
.character_15	01111010	Variable	Internal
.character_16	00000000	Variable	Internal
.serial	00000000	Variable	Internal
.refresh	00000000000000000101...	Variable	Internal
.retry	00000000000000000000...	Variable	Internal
.expire	00000000000000100100...	Variable	Internal
.ttl	00000001000000000101...	Variable	Internal

Figures 4.22.b : Résultat de simulation du module SOA (fenêtre structure)

4. 3.2.3 Module MUX_OUT

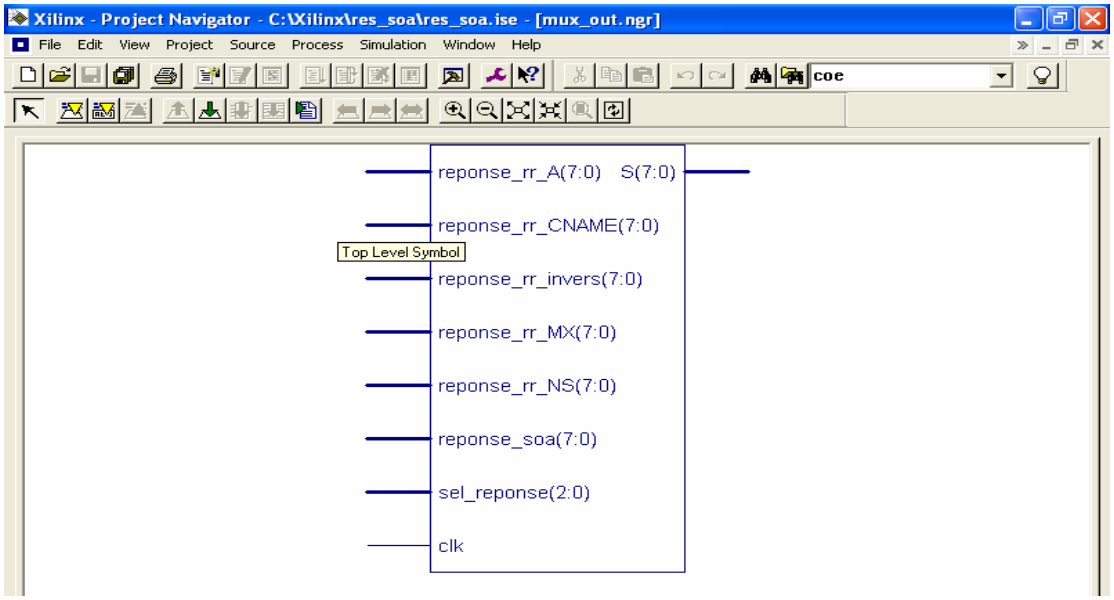


Figure 4.23 : Vue externe du module *MUX_OUT*

Le Tableau 5 donne les affectations effectuées sur les ports de sortie du multiplexeur selon la valeur de *sel_reponse*

Valeur de sel_reponse	ports de sortie correspondant à la valeur de sel_reponse
« 000 »	reponse_rr_invers
« 001 »	reponse_rr_A
« 010 »	reponse_soa
« 011 »	reponse_rr_NS
« 100»	reponse_rr_MX
« 101»	reponse_rr_CNAME

Tableau 5 : Valeurs de signaux de sortie du multiplexeur en fonction du port *sel_reponse*

4.3.3 Bloc Resolution

La figure ci après résume et illustre l'algorithme de fonctionnement du bloc *resolution* (le noyau du serveur DNS).

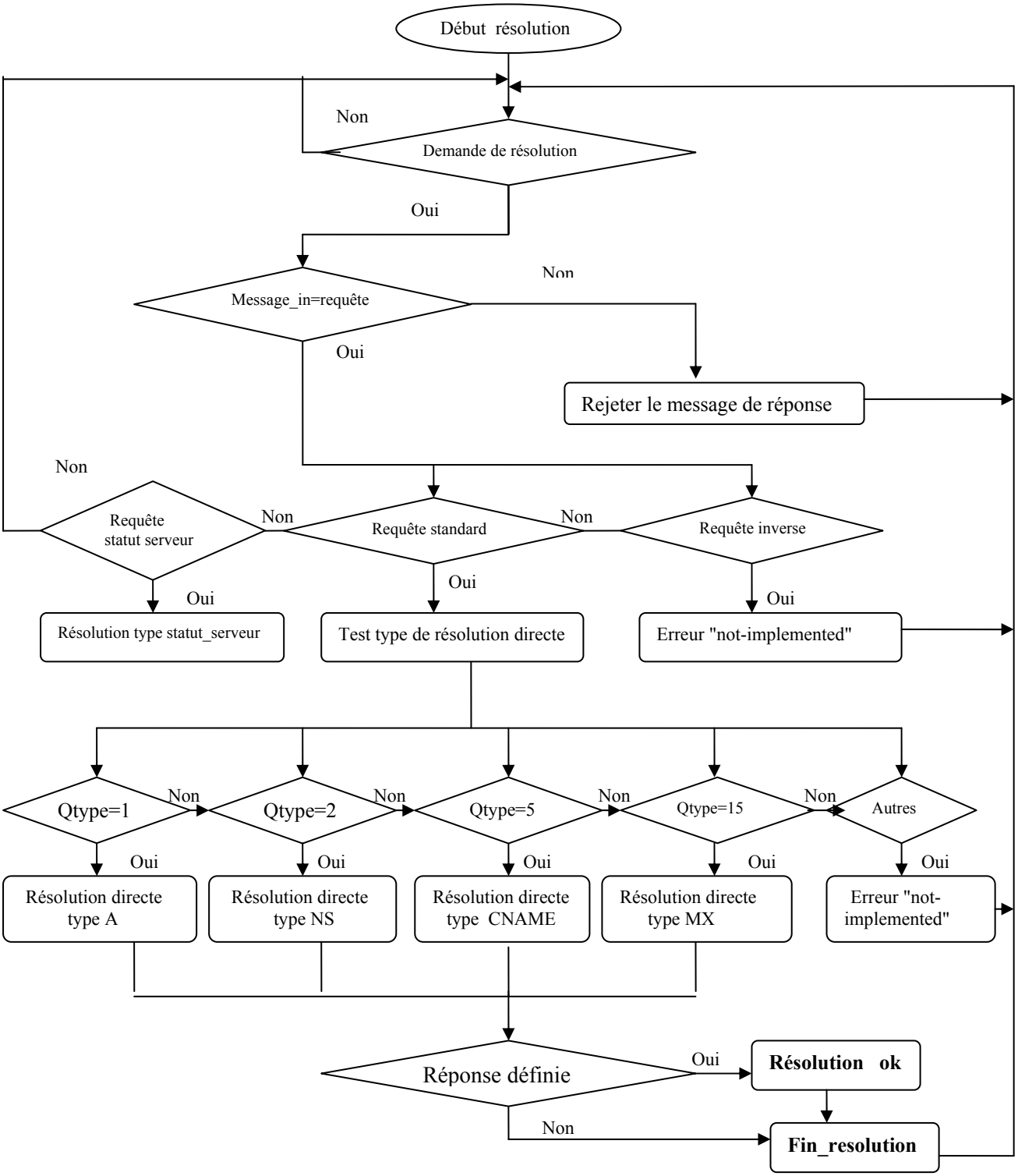


Figure 4.24 : Organigramme du bloc *resolution*

4. 3.3.1 Description fonctionnelle

C'est le noyau de l'architecture proposée pour l'implémentation du serveur DNS autoritaire. Il exécute un nombre de fonctions de résolution ; la principale étant la résolution directe de nom.

La réponse fournie par ce bloc est dépendante des informations fournies par le bloc *reception*. Entre autre le type de requête, sa classe et bien sûre le nom de domaine sur le quel le serveur DNS est interrogé. Pour répondre aux différentes requêtes, ce bloc utilise les RRs stockés dans le sous bloc données_RRs du bloc configuration (cette partie de la configuration est représentée sur la figure ci après. Notons que le sous bloc donnée_RRs n'est pas inclus dans le bloc *resolution*, il n'est représenté que pour mieux illustrer son fonctionnement).

La structure interne du bloc *resolution* est comme suit :

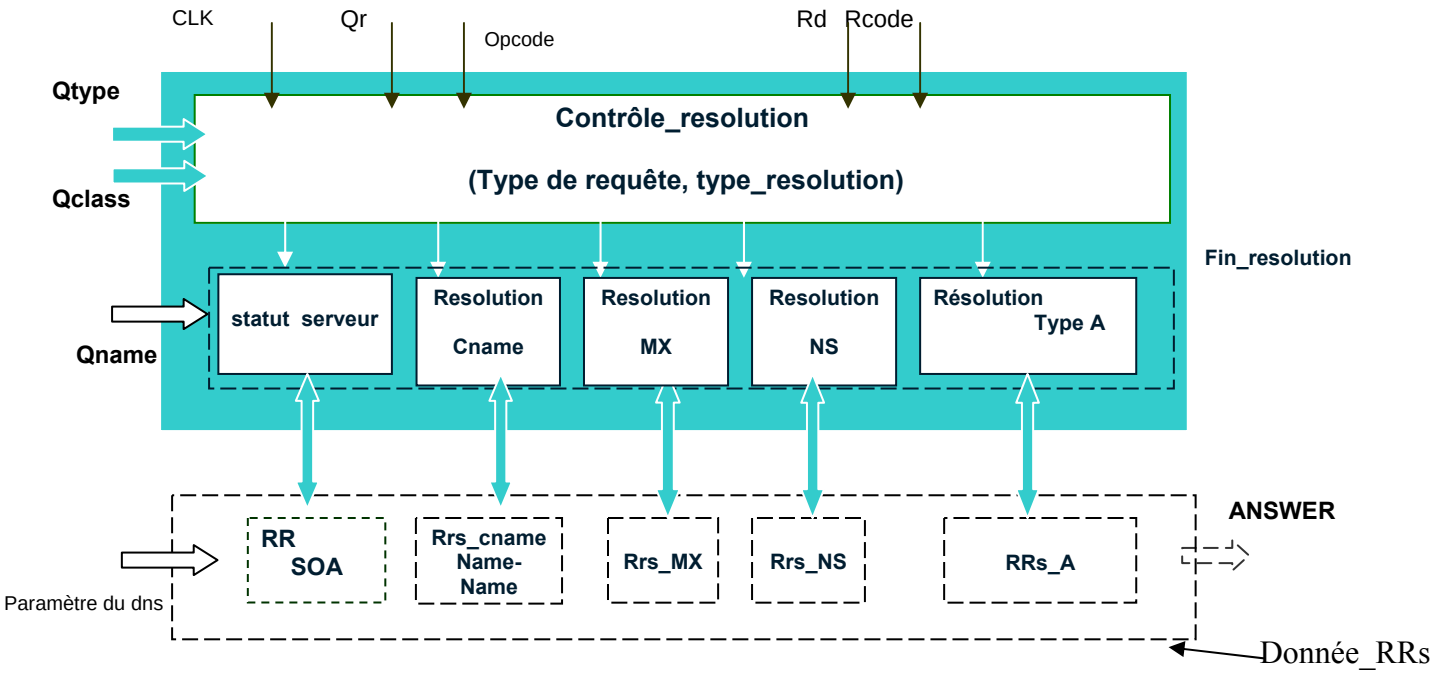


Figure4.25 : Architecture interne du bloc résolution

Comme le montre la figure ci-dessus, le bloc résolution comporte deux sous bloc :

4. 3.3.2 Sous bloc type de résolution

Ce sous bloc comporte cinq modules, séparément chaque module effectue le type de résolution correspondant à la requête après réception d'un signal de sélection « *resol_in* » en provenance du module *control_resolution*. Le principe de fonctionnement est le même pour chaque module de résolution. La seule différence est dans le type de donnée recherchée. Par rapport au type de la requête, le module de résolution sélectionné cherchera le RR de réponse correspondant à Qname dans les mémoires du bloc donnée_RRs, selon l'information demandée par le requérant.

Nous détaillerons dans ce qui suit le fonctionnement du module résolution directe –type A-

4. 3.3.2.1 Module résolution directe

Ce module est un circuit dont la principale tâche est la comparaison de qname en entrée et le champ name de chaque RRs du module mémoire RRs correspondant au type spécifié dans la requête. Il sélectionne le RR de type A contenant l'adresse IP recherchée afin de l'envoyer en réponse. Le type rrs_type_a est déclaré dans un *package resolution* au quel l'ensemble des modules du bloc *resolution* fera référence. Ce module reçois le bit *Resol_in* égale à '1' indiquant le début d'une demande de résolution directe (ce signal reste à '1' tout le temps de la résolution).

Dès que ce module achève son traitement (assigner l'adresse IP correspondante à qname), le bit *fin_resol* passe à '1' indiquant la fin de résolution directe, le module contrôle résolution envoie au multiplexeur son état de traitement, en mettant le champ *sel_reponse* à '1' (les affectations effectuées sur le port de sortie du multiplexeur est selon la valeur de *sel_reponse*). Selon cet état, le module *mux_out* sélectionnera parmi les sorties des modules donnés RRs ; le port de sortie associée au module contenant le RR recherché. La sortie du multiplexeur véhicule le RR réponse correspondant à la question. Cette sortie sera l'entrée du module answer du bloc *emission*.

Module résolution NS le rôle de ce module comme pour le module résolution directe type A est d'assigner une adresse IP à un nom mais dans ce cas il s'agie d'une adresse

IP du serveur de noms "autorisé" pour le domaine recherché si le la donnée recherchée n'est pas d'autorité.

4. 3. 3.2.2 Module résolution Cname

Son principe de fonctionnement est identique à celui du bloc résolution directe type A. La différence par rapport à la résolution directe type A est que le champ RDATA n'est plus une adresse IP mais un nom canonique d'un alias de nom donc un autre nom d'hôte.

Ce module assigne un nom canonique au *Qname* de la requête.

4. 3. 3.2.3 Module statut serveur

Son rôle est de chercher les paramètres du champ SOA, le début d'une zone d'autorité, pour les envoyer en réponse au serveur de nom requérant.

4.3.3.3 Sous bloc contrôle_résolution

C'est l'unité responsable de la gestion et de l'analyse des données transférées au bloc *resolution*. Elle génère l'ensemble des signaux de commande allant vers les modules du sous bloc type résolution et aussi quelques signaux vers le bloc *emission,* essentiellement le signal *reponse* correspondant à un ordre d'émission après la fin de la résolution.

Son rôle est aussi d'interpréter les résultats en provenance du bloc *control_reception* et de détecter selon les données extraites du bloc *reception* le type du message reçus. Dans le cas d'une réponse, le message de réponse sera rejeté et le traitement s'arrête. Dans le cas contraire (requête DNS), le bloc *contrôle_resolution* déterminera le type de résolution demandée en traitant la donnée *Qtype*. Suivant la valeur de ce champ, un des module du sous bloc résolution sera sélectionné. Le bloc *contrôle_resolution* attend le résultat du traitement du module résolution et envoie des signaux de commande au bloc *emission* pour que ce dernier puisse effectuer son rôle de transmetteur de la réponse au client après composition du *message_dns_out*.

4. 3.3 .4 Simulation bloc resolution

4. 3.3.4 .1 Simulation du sous bloc type de résolution

Dans cette section, nous allons donner les résultats de simulation pour les modules composant le sous bloc type resolution et ceci pour deux descriptions différentes. L'une est une description pour des modules non synthétisable permettant d'illustrer le mécanisme de *resolution* et de bien visualiser la translation en VHDL du processus de résolution en montrant les différents champs de la réponse à une requête DNS pour différents types d'enregistrements.

En effet, les premières description faites pour les modules de résolution sont des descriptions pouvant être simulées au niveau comportementale seulement et contenant des types non synthétisable. Ces descriptions ont été faites dont le but de modéliser le processus de résolution.

C'est à partir des secondes descriptions synthétisables et de leurs interconnections que nous avons obtenu le schéma de la structure interne du bloc *resolution* représentée par la figure 4.42 .Nous donnerons en premier lieu les tests pour les premières descriptions qui illustrent mieux la fonctionnalité des modules de résolution ; à savoir le module résolution directe, Cname et NS. Par la suite nous donnerons le résultas pour la deuxième description et aussi la vus externe pour chaque module du sous bloc type résolution.

Remarque : nous suffixons les modules de résolution pour les premières descriptions en ajoutant (D1) et (D2) pour les secondes descriptions.

A. Test du module Resolution_directe

A-1-a Résolution directe type A (D1)

Pour le test de ce module, nous avons passé qname « serveur.enp.edu.dz » en générique et nous avons créé dans le corps de l'architecture de ce module un tableau nommé *tableau_rrs_a_type* de RRs de type A (sa structure est illustrée dans la fenêtre structure).

Le type *rrs_type_a* est déclaré dans un package *résolution* auquel l'ensemble des modules du bloc *resolution* fera référencé.

Ce tableau de test contiendra une liste de RRs de même type (le champ RDATA est une adresse IP de quatre octets pour tout les RRs de ce tableau). C'est dans cette liste que le module résolution directe cherchera la bonne adresse IP et le RR à envoyer.

Ce tableau émule donc la mémoire *RRs_type_a* qui sert à stocker ces RRs et qui sera modélisé par le module données_RRs.

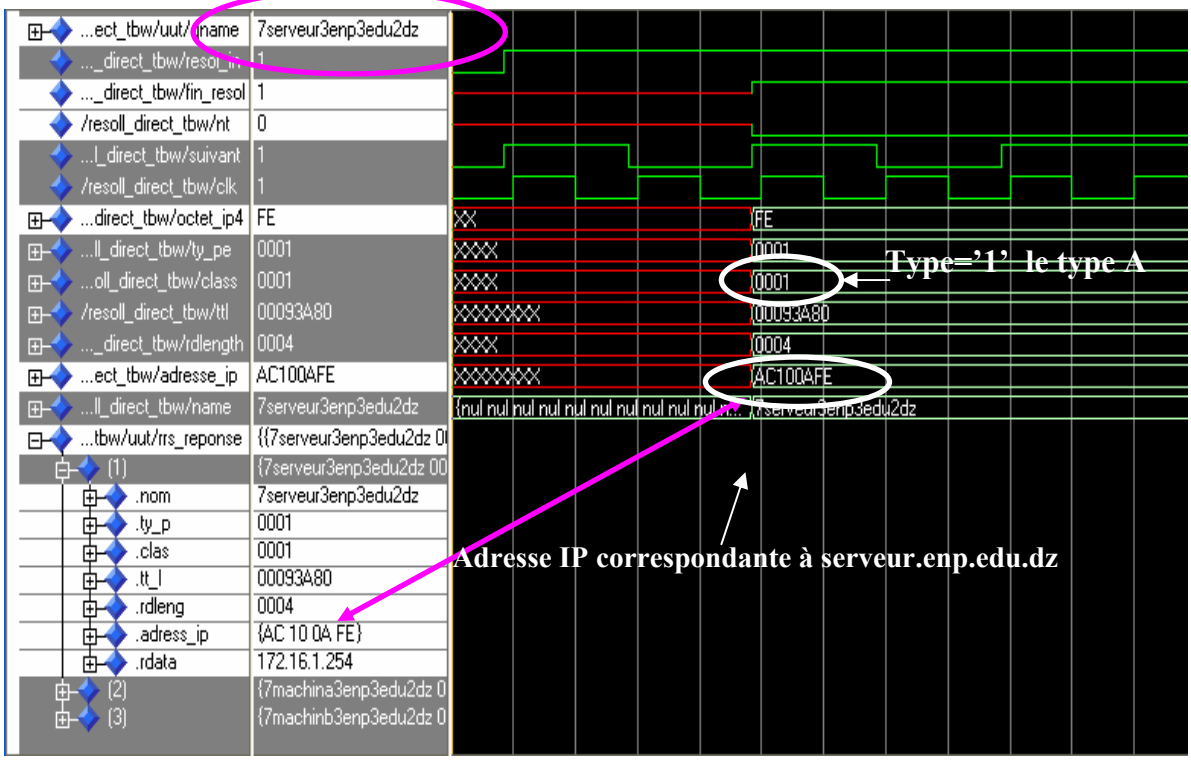
On simule alors l'action du module *type_requete* en forçant le bit *Resol_in* indiquant le début d'une demande de résolution directe à '1'.

Dès que ce module achève son traitement (assigne l'adresse IP correspondante à qname), le bit *fin_resol_direct* passe à '1' indiquant la fin de résolution directe.

En remarque que dans le cas où *resol_in* est à '1' et *caract_suiv* est actif sur front montant, l'affectation des sorties ne se fait pas. Elle se fera au prochain front montant de *caract_suivant*. En remarque alors qu'au moment du front montant de *caract_suivant*, les valeurs des champs du RR recherchées sont affectées aux sorties correspondantes.

Le résultat de simulation est comme suit :

1-La réponse à qname = serveur.enp.edu.dz



Figures 4.26.a : Résultat de simulation du module résolution type A (D1)

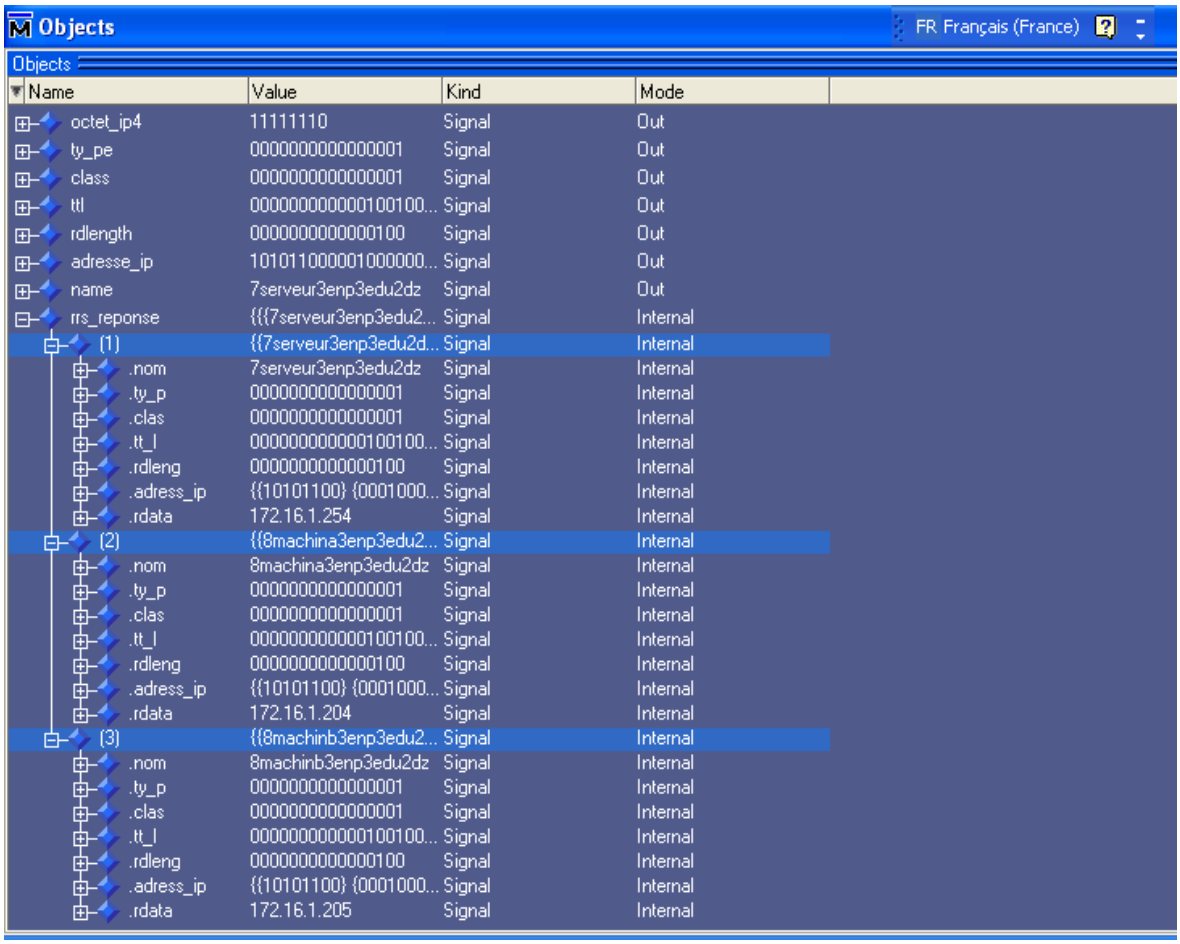
On reconnaît bien en sortie **adresse_ip** ; la chaîne de quatre octets correspondant à L'adresse de la machine recherchée

On remarque aussi que le champ TTL prend la valeur donnée par le RR dans la liste rrs_reponse, de même pour le champ rdlength .On vérifie ainsi chaque champ du RR de la réponse à envoyer l'une après l'autre. La réponse est l'enregistrement de ressource de type A correspondant au RR recherché par qname parmi un ensemble de RRs de type A.

Le choix est donc fait pour le RR correspondant bien à qname.

Remarque : la sortie octet_ip4 n'est pas incluse dans l'architecture de ce module. Elle est ajoutée pour illustrer le quatrième octet de l'adresse IP et valider ainsi le résultat.

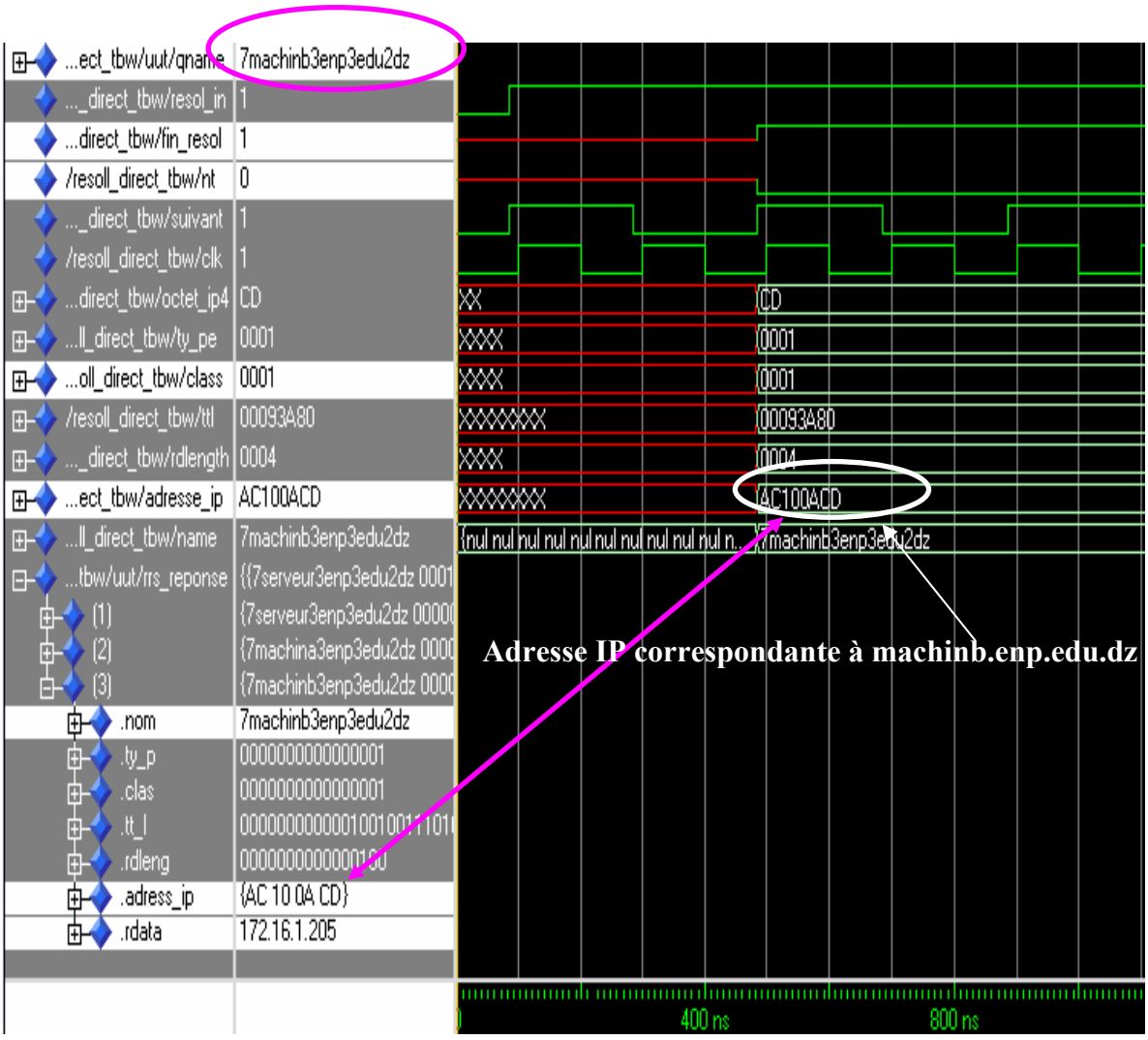
La fenêtre structure pour le premier test est donnée ci après



Figures 4.26.b : Résultat de simulation du module résolution type A (D1 fenêtre structure)

Pour vérifier que ce module traite bien les demandes de résolution qui lui sont destinées, on modifie le qname par un autre nom sur lequel portera la nouvelle recherche de l'adresse IP. On obtient le résultat ci après, on remarque bien que pour ce nouveau qname, l'adresse_ip est changée par celle correspondant au nouveau qname. On vérifie aussi que le RRs envoyé en sortie est celui recherché.

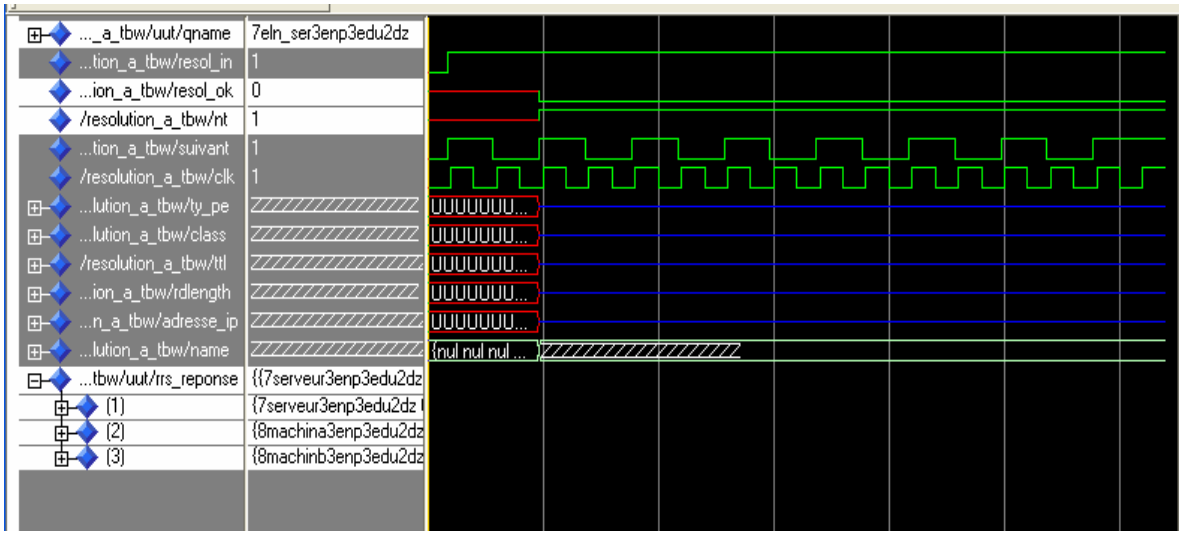
2-La réponse à qname =machinb.enp.edu.dz



Figures 4.26.c: Résultat de simulation du module résolution type A (D1)

Dans le cas ou le nom n'est pas trouvé

Comme montré ci après dans le cas où le RR recherché n'est pas trouvé, le bit *NT* (non trouvé) passe à '1' indiquant que le nom pour le quelle en cherche l'adresse IP n'est pas trouvé. Ce bit (quant il est à '1') sera interprété par le bloc *contrôle_résolution* par le forçage du champ *Rcode* à '3' et *Rcode* à '0' quant ce bit est à '0'. Aussi, le bit *resol_ok* passe à '0'.



Figures 4.26.d : Résultat de simulation du module résolution type A (D1)

A-1-b Résolution directe type A (D2) Vue externe de ce module

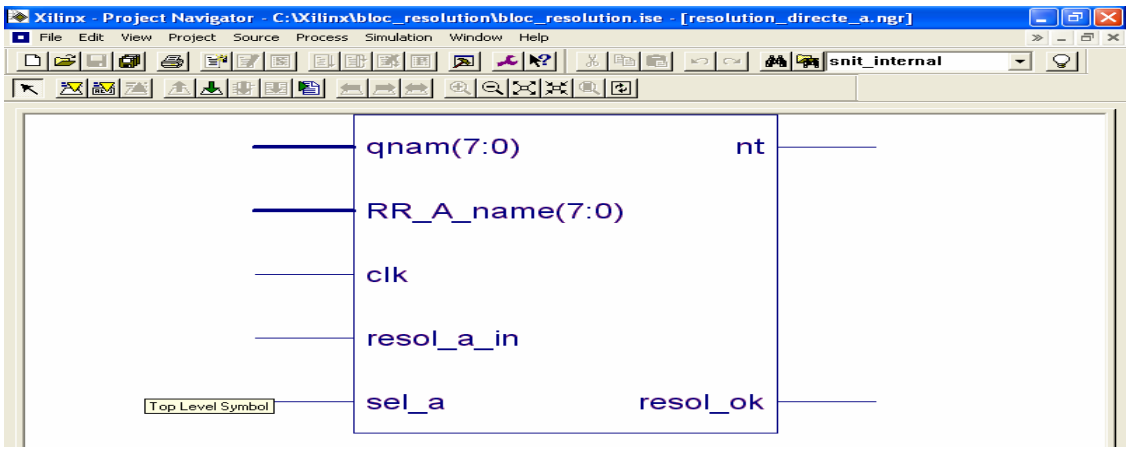
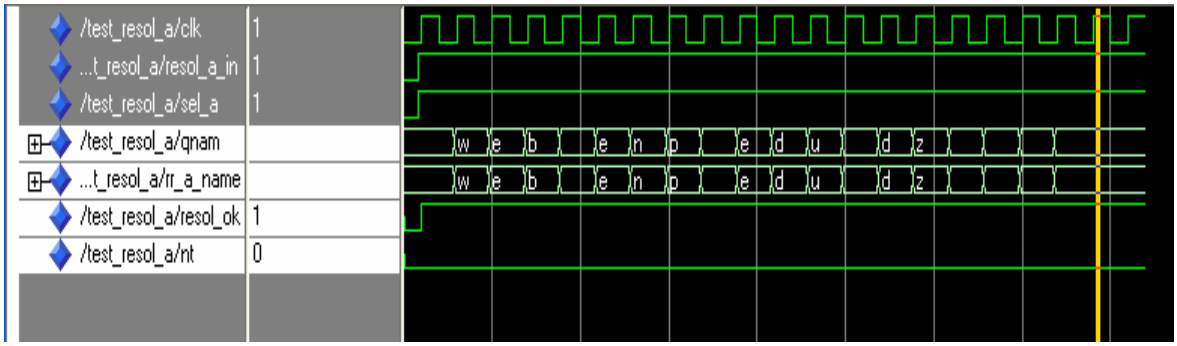


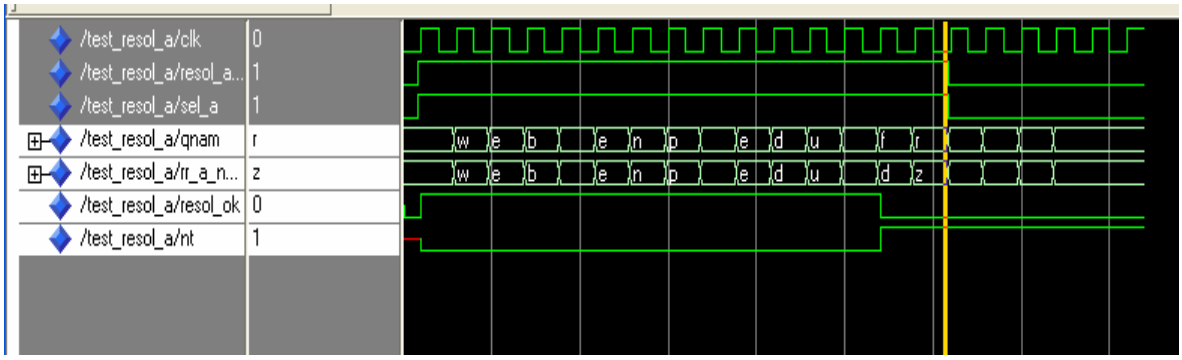
Figure 4.27 : Vue externe du module *resolution directe type A*

Simulation



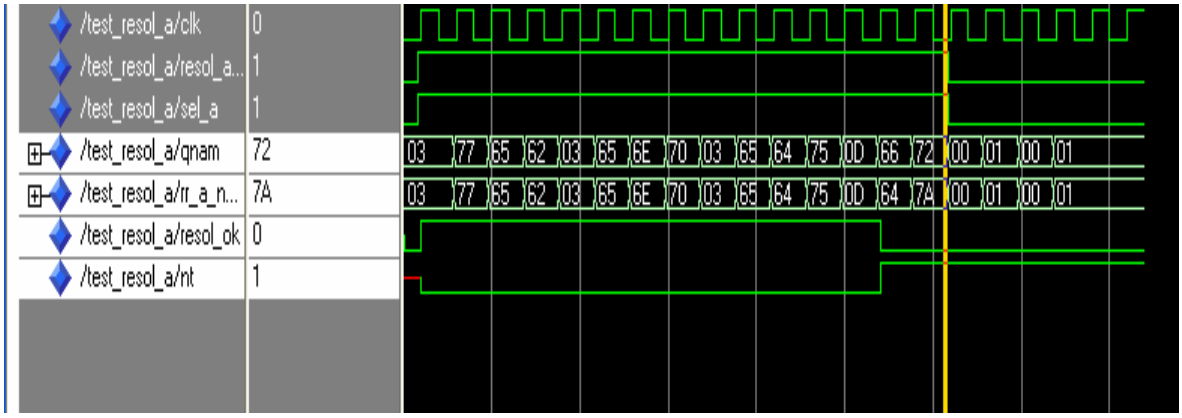
Figures 4.28.a : Résultat de simulation du module résolution type A (D2)

Dans le cas où le nom n'est pas trouvé



Figures 4.28.b : Résultat de simulation du module résolution type A (D2)

Ci après une représentation en hexa des octets comparés



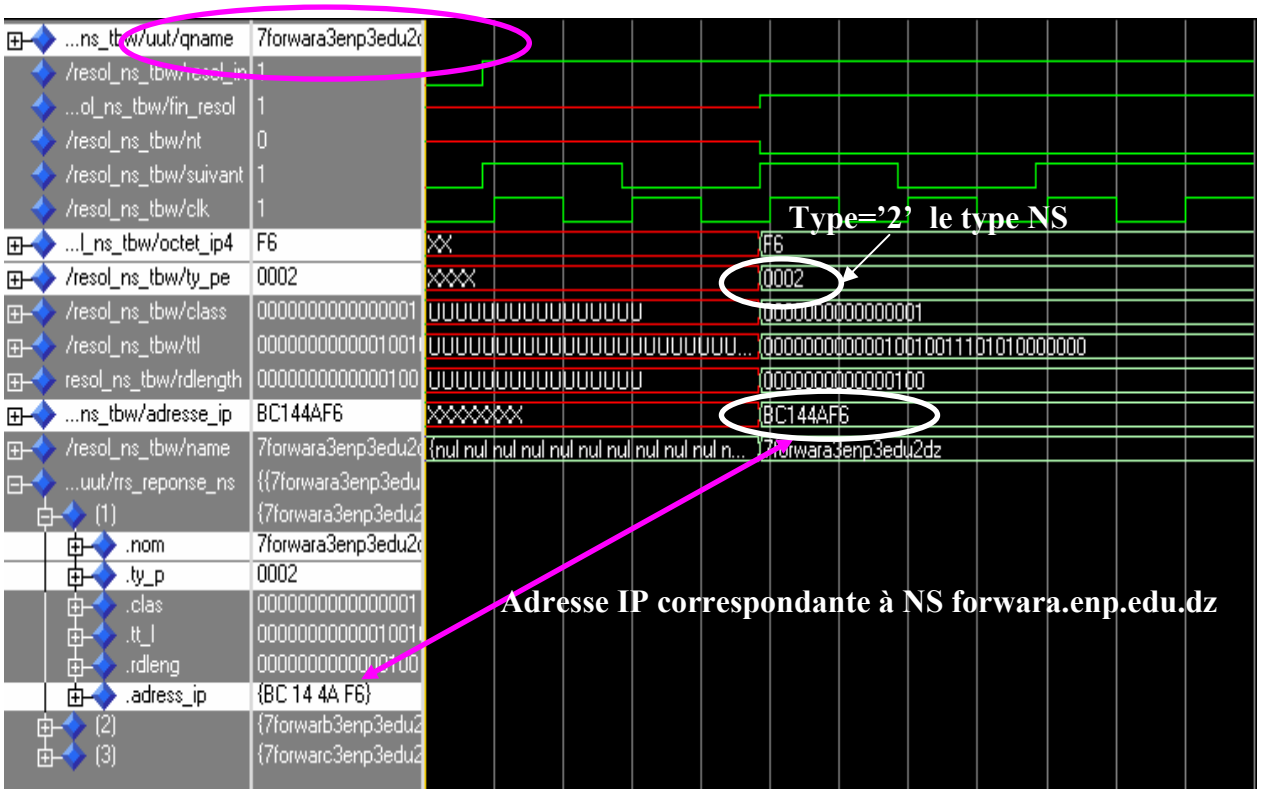
Figures 4.28.c : Résultat de simulation du module résolution type A (D2)

Comme nous pouvons voir sur les figures ci-dessus la comparaison se fait octet par octet jusqu' a la fin de qname .Dans le cas où tout les octet sont identique le bit *resol_ok* passe à '1' ce qui signifie que le nom est trouvé dans le cas contraire c'est le bit *NT*

(non trouvé) qui passe à '1'comme le montre la figure 4.28.c (si un octet des deux chaînes de caractère à comparer diffère le bit *NT* passe à '1')

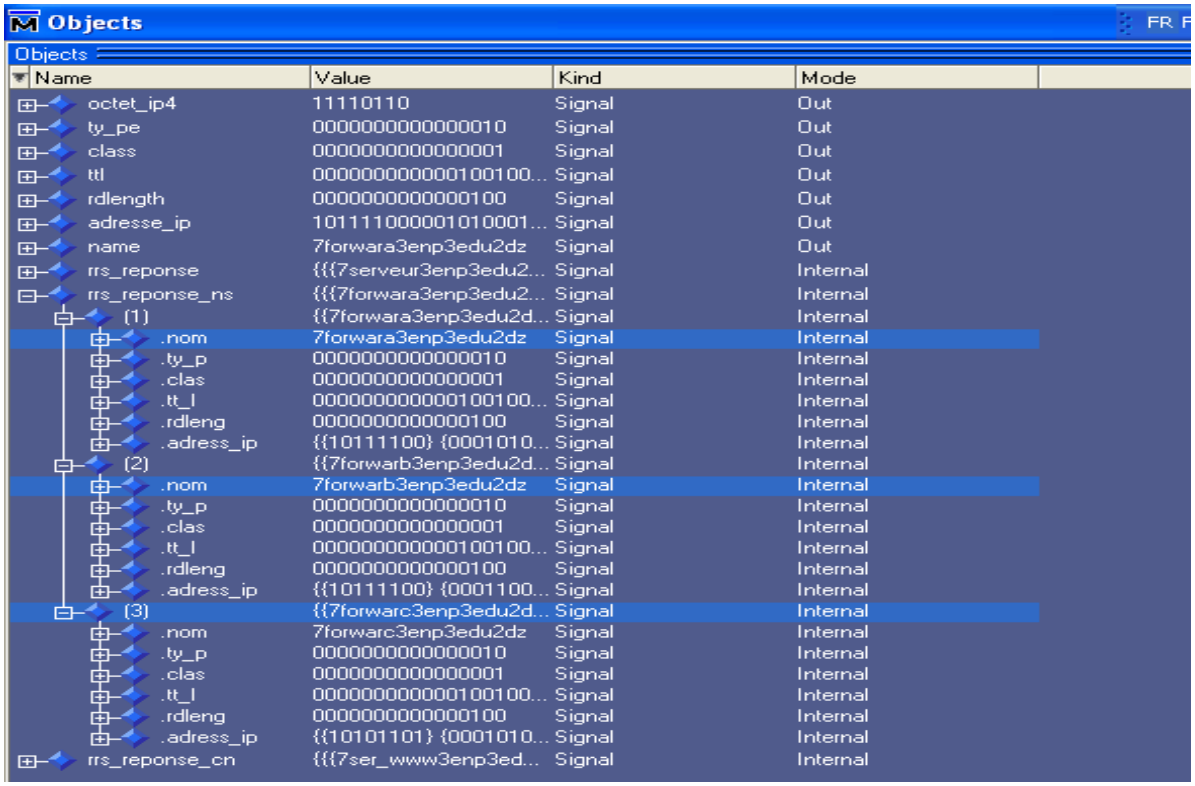
Notons que c'est le même principe pour tous les autres modules de résolution de la seconde description (D2).

A-2-a Résolution directe type NS (D1)



Figures 4.29.a : Résultat de simulation du module résolution type NS (D1)

Sur la fenêtre structure, on peut voir la liste dans laquelle le RR type NS (le champ *type* dans ce cas est égale à '2') recherché est extrait.



Figures 4.29.b : Résultat de simulation du module résolution type NS (D1 fenêtre structure)

A-1-b Résolution type NS (D2)

Vue externe

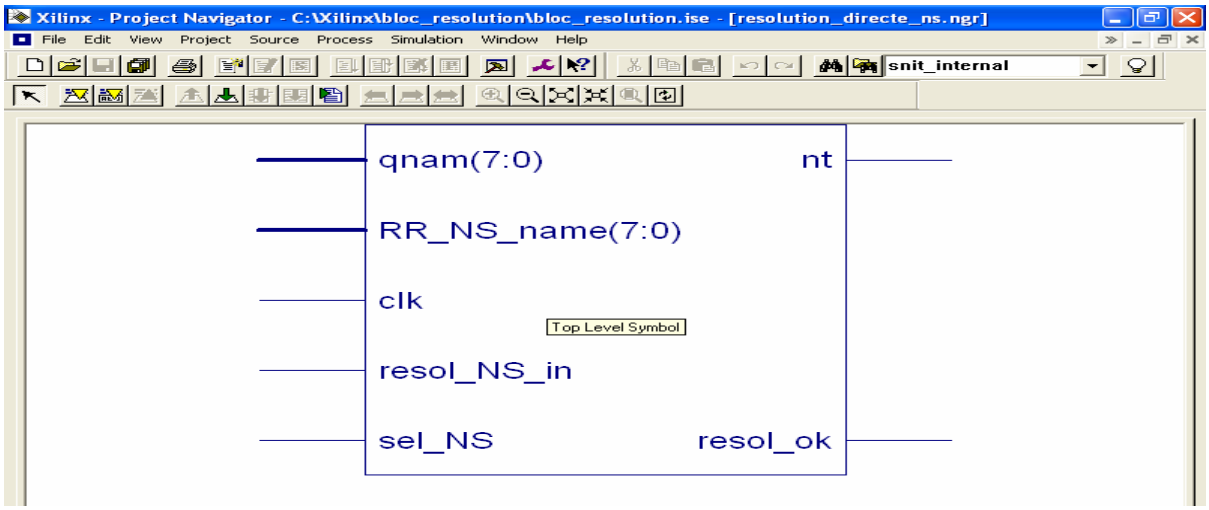


Figure 4.30 : Vue externe du module résolution type NS (D2)

Simulation

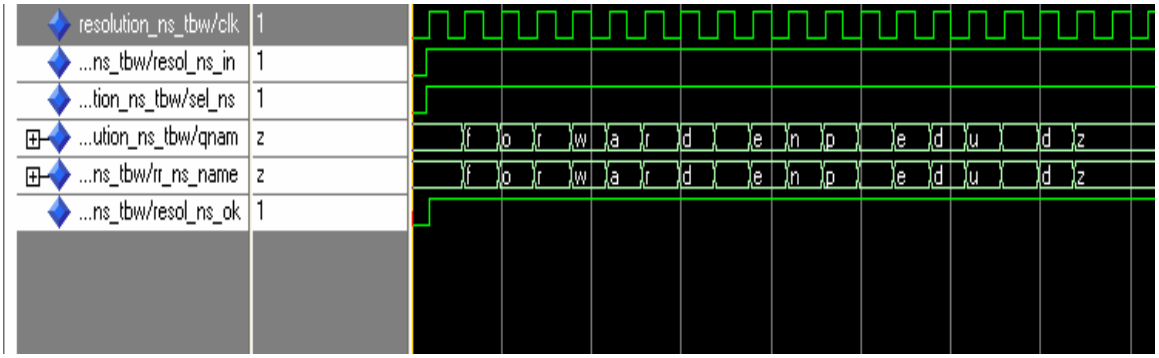


Figure 4.31.a : Résultat de simulation du module résolution type NS (D2)

Ci après une représentation en hexa des octets comparés

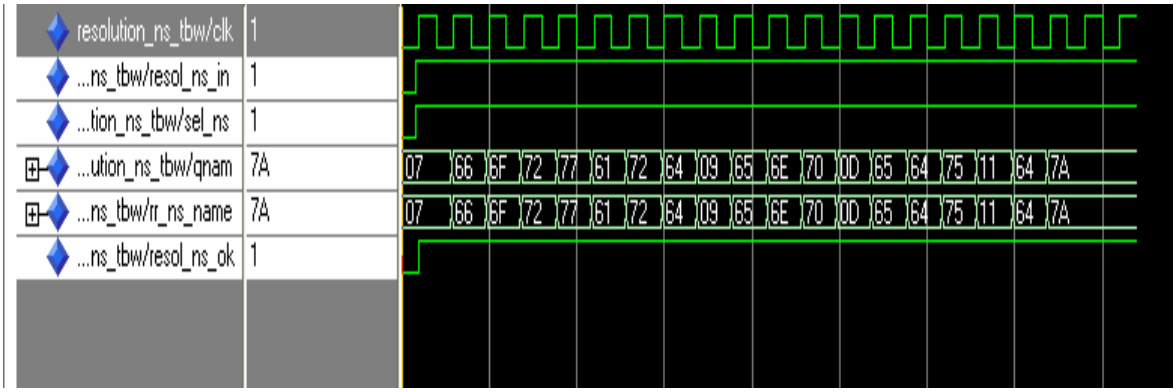


Figure 4.31.b : Résultat de simulation du module résolution type NS (D2)

Dans le cas ou le RR NS n'est pas trouvé

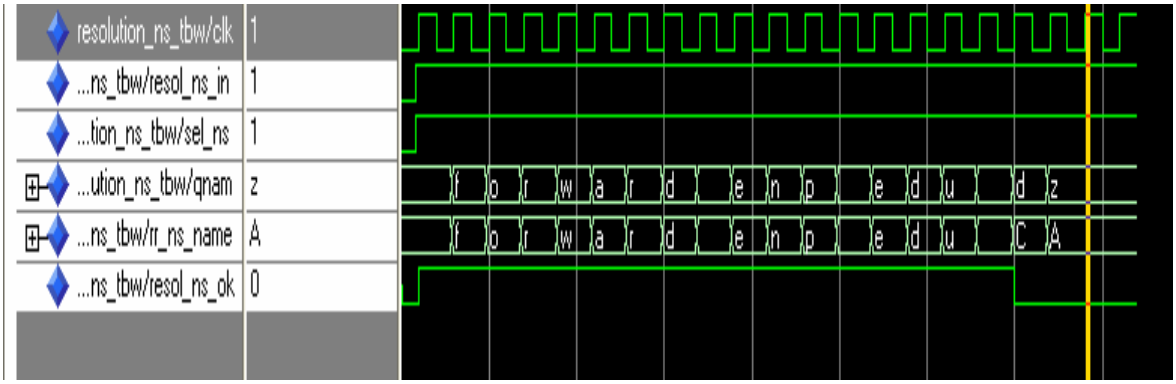
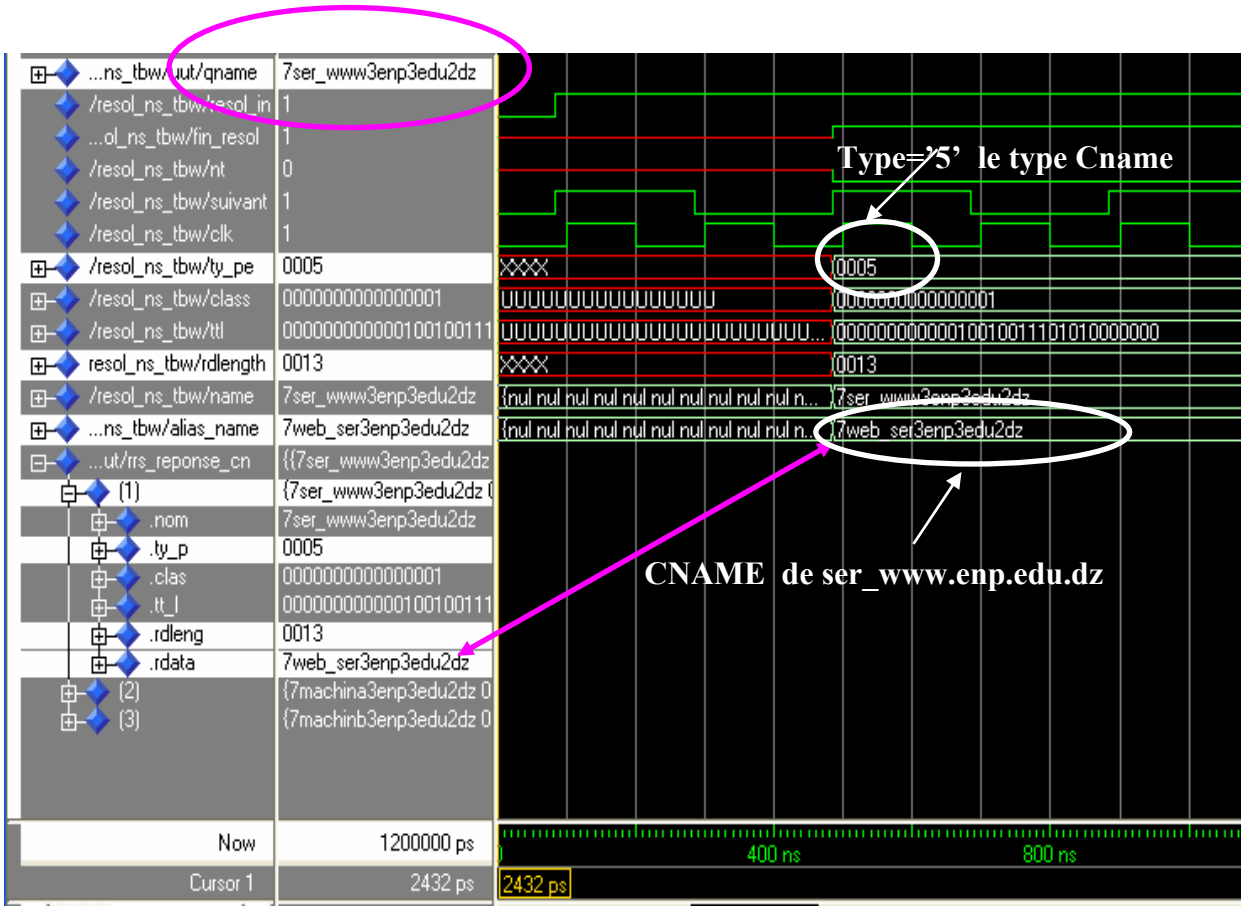


Figure 4.31.c : Résultat de simulation du module résolution type NS (D2)

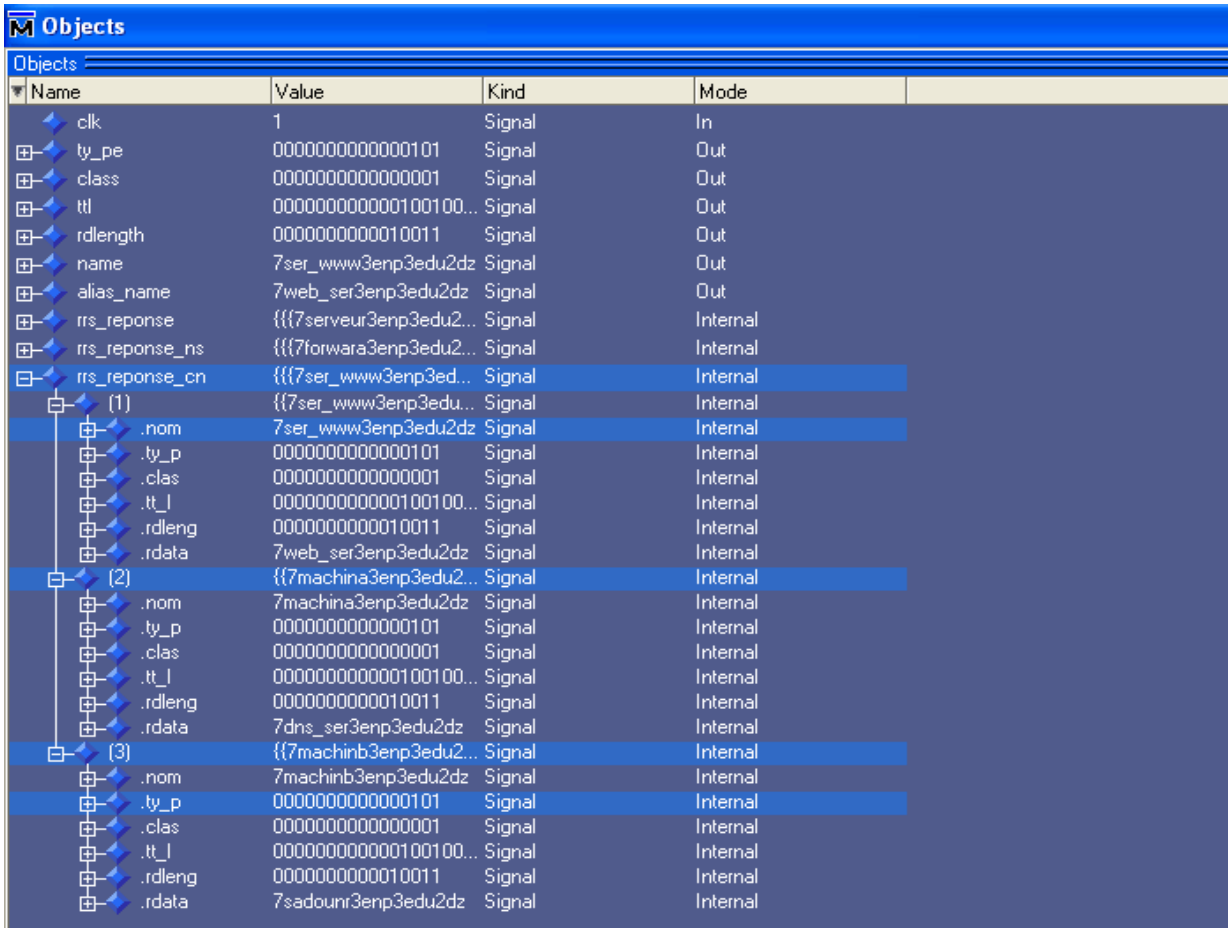
A-3-a Résolution directe type CNAME (D1)

La différence par rapport à la résolution directe type A est que le champ RDATA n'est plus une adresse IP mais un alias de nom. On a donc un autre nom d'hôte, comme le montre le résultat de simulation du module résolution type Cname, dans ce cas le champ *type* est égal à '5'.



Figures 4.32.a : Résultat de simulation du module résolution type CNAME (D1)

De même, sur la fenêtre structure on peut voir la liste dans laquelle le RR type Cname (le champ *type* dans ce cas est égale à '5') recherché est extrait.



Figures 4.32.b : Résultat de simulation du module résolution type CNAME (D1 fenêtre structure)

A-3-b Résolution type CNAME (D2)

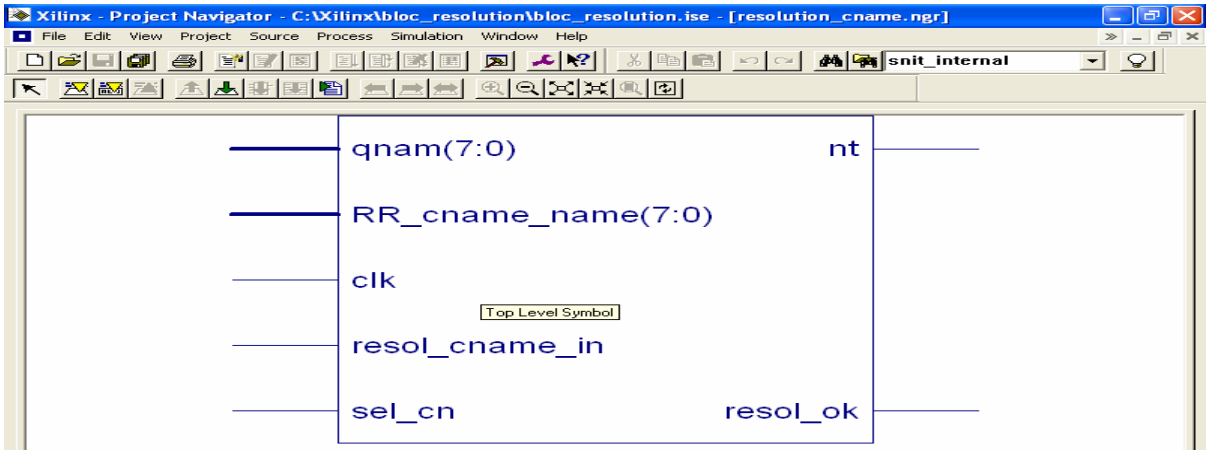


Figure 4.33 : Vue externe du module resolution type CNAME (D2)

Simulation

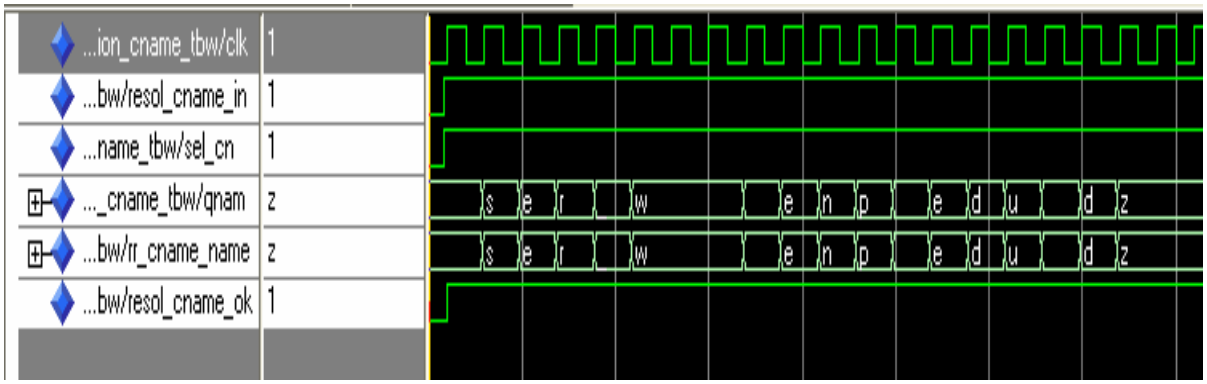


Figure 4.34.a : Résultat de simulation du module résolution type CNAME (D2)

Dans le cas où la le nom canonique n'est pas trouvé

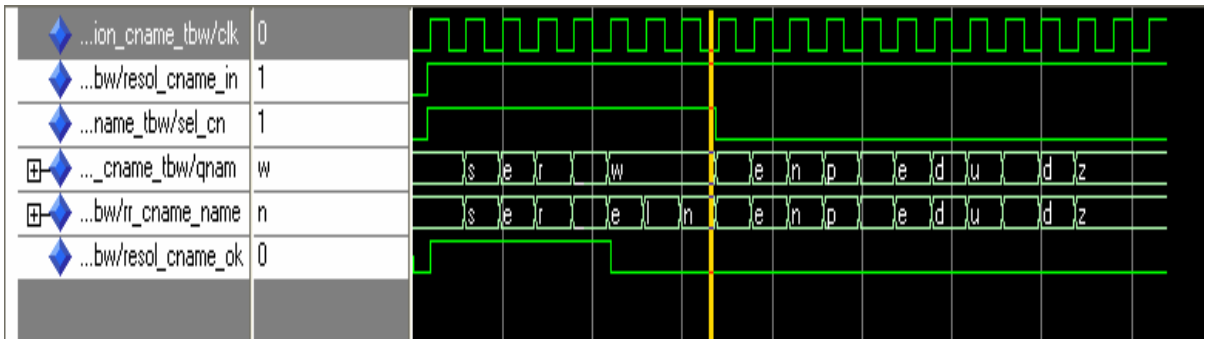


Figure 4.34.b : Résultat de simulation du module résolution type CNAME (D2)

Ci après une représentation en hexa des octets comparés

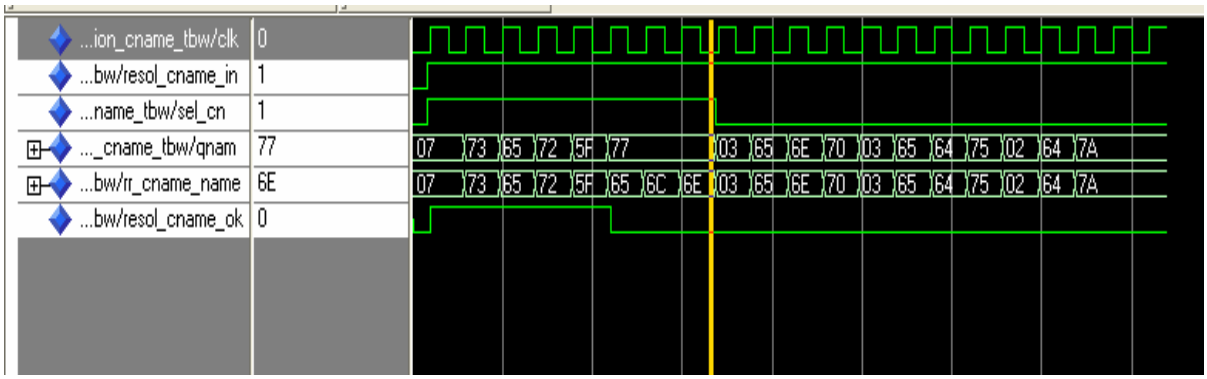


Figure 4.34.c : Résultat de simulation du module résolution type CNAME (D2)

A-4 Module statut serveur

La vue externe du module statut serveur est comme suit

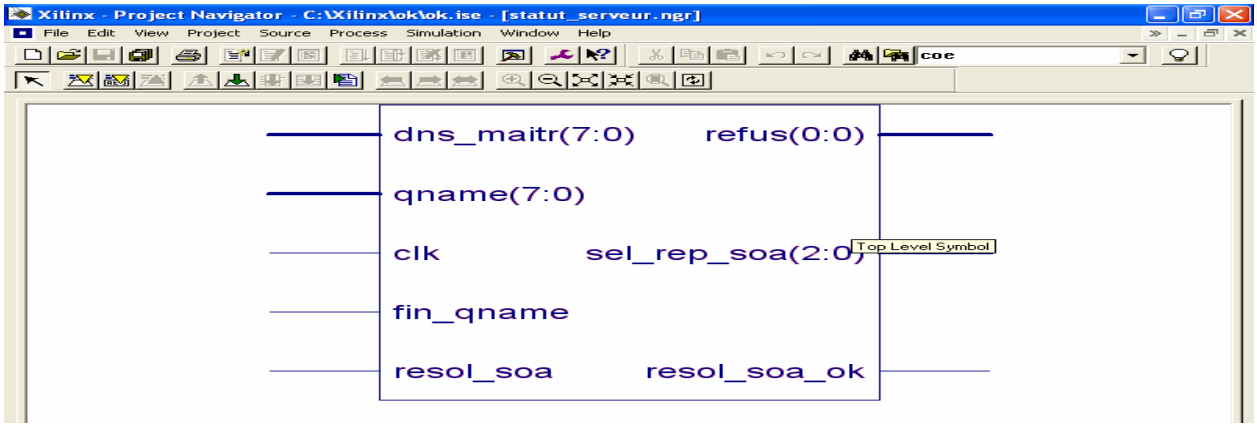


Figure 4.35 : Vue externe du module statut serveur

Résultats de simulation

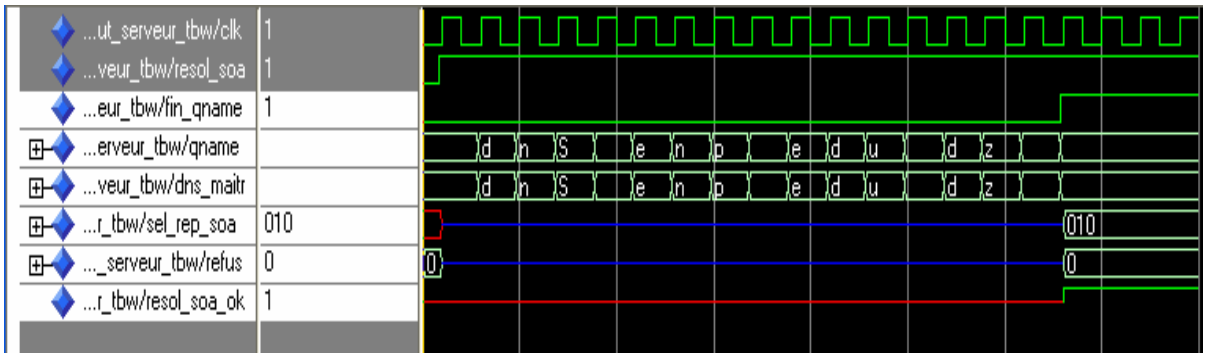


Figure 4.36.a : Résultat de simulation du module statut serveur

Dans le cas ou le dns_maitre est déferant de celui de la requête on obtient le résultat ci après.

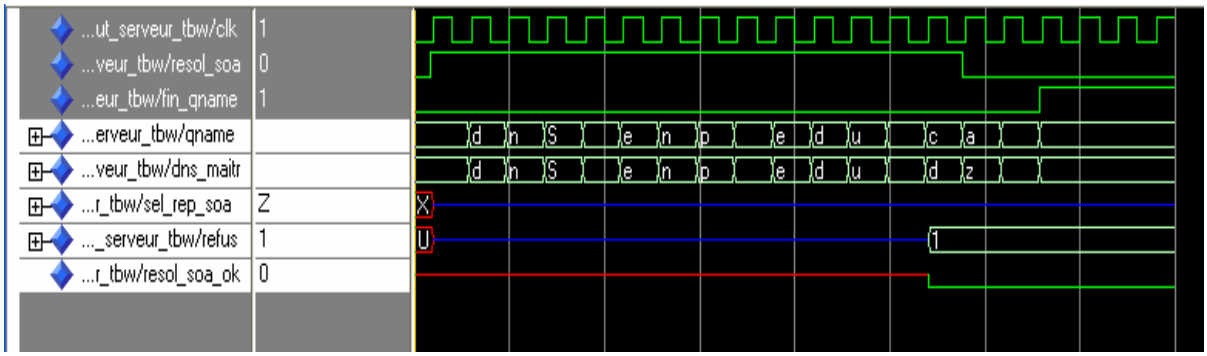


Figure 4.36.b : Résultat de simulation du module statut serveur

On peut remarquer que si la sortie *refus* est à '1', le serveur DNS maître refuse d'envoyer les paramètres du champ SOA au serveur de nom requérant. Dans le cas contraire, le serveur maître envoie ce RR au serveur esclave.

Le champ *sel_rep_soa* de trois bits sera égal à '2' dans le cas d'accord, cette sortie sera l'entrée de sélection du *mux_out* du bloc *configuration*. Ce dernier sélectionnera la mémoire de données RRs correspondante, et la sortie sera celle du module SOA. Il sera envoyé au module *answer* du bloc *emission* en vue de composer le message de réponse à envoyer au client. Dans le cas contraire, cette sortie sera en haute impédance comme on peut le remarquer sur la figure 4.36.b (dans le cas de refus).

4. 3.3.4.2 Simulation sous bloc contrôle résolution

1- Machine d'état *contrôle_resolution*

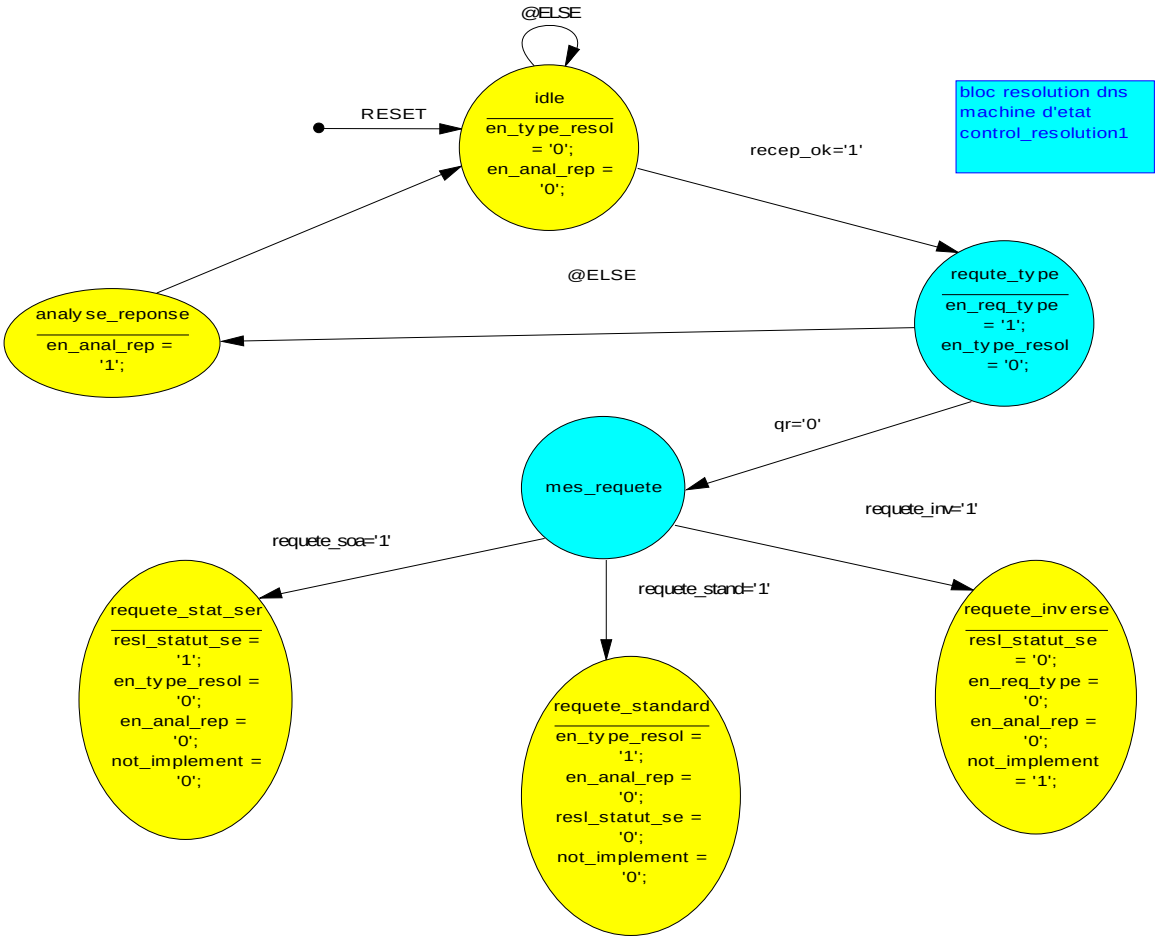
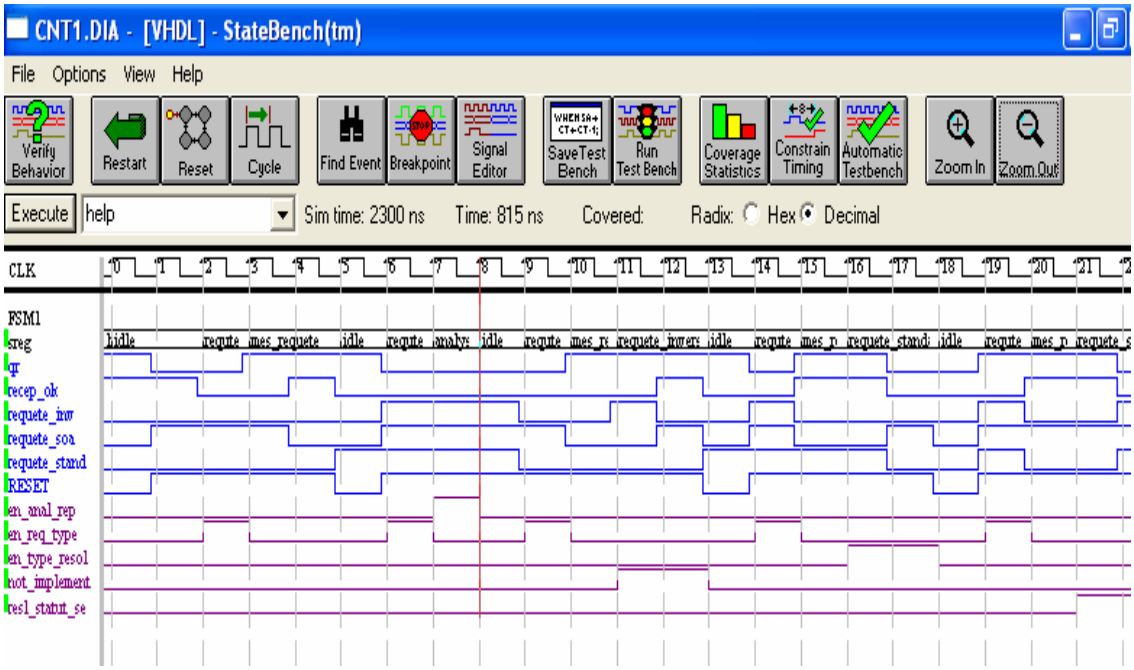
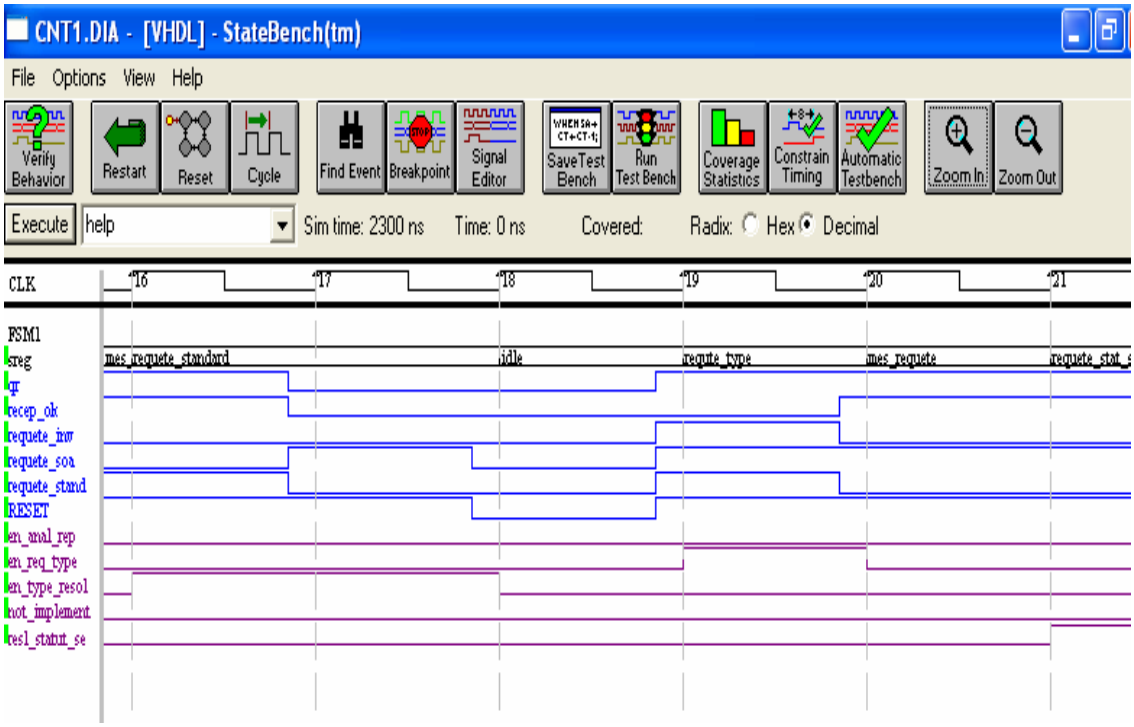


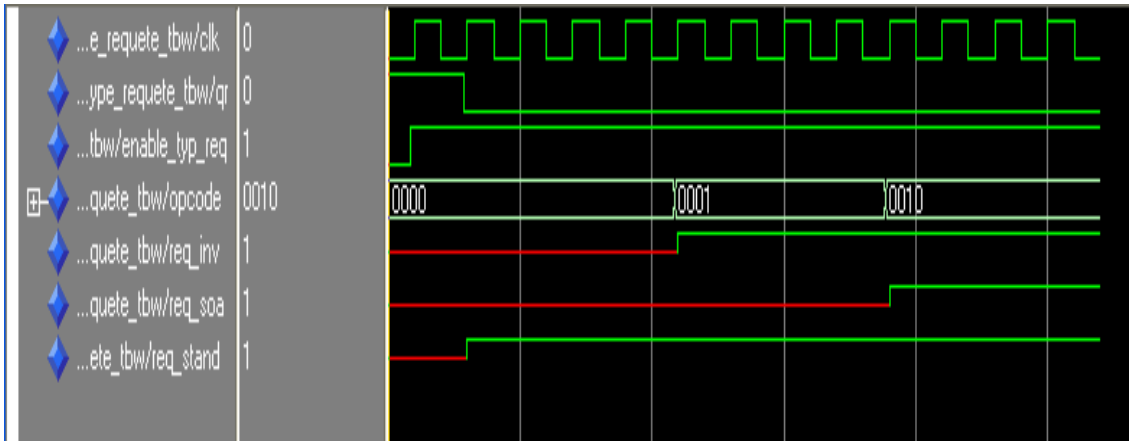
Figure 4.37 : Machine à état du module *contrôle_resolution*

Ci après les résultats de simulation pour les modules de contrôle au niveau du bloc *resolution*

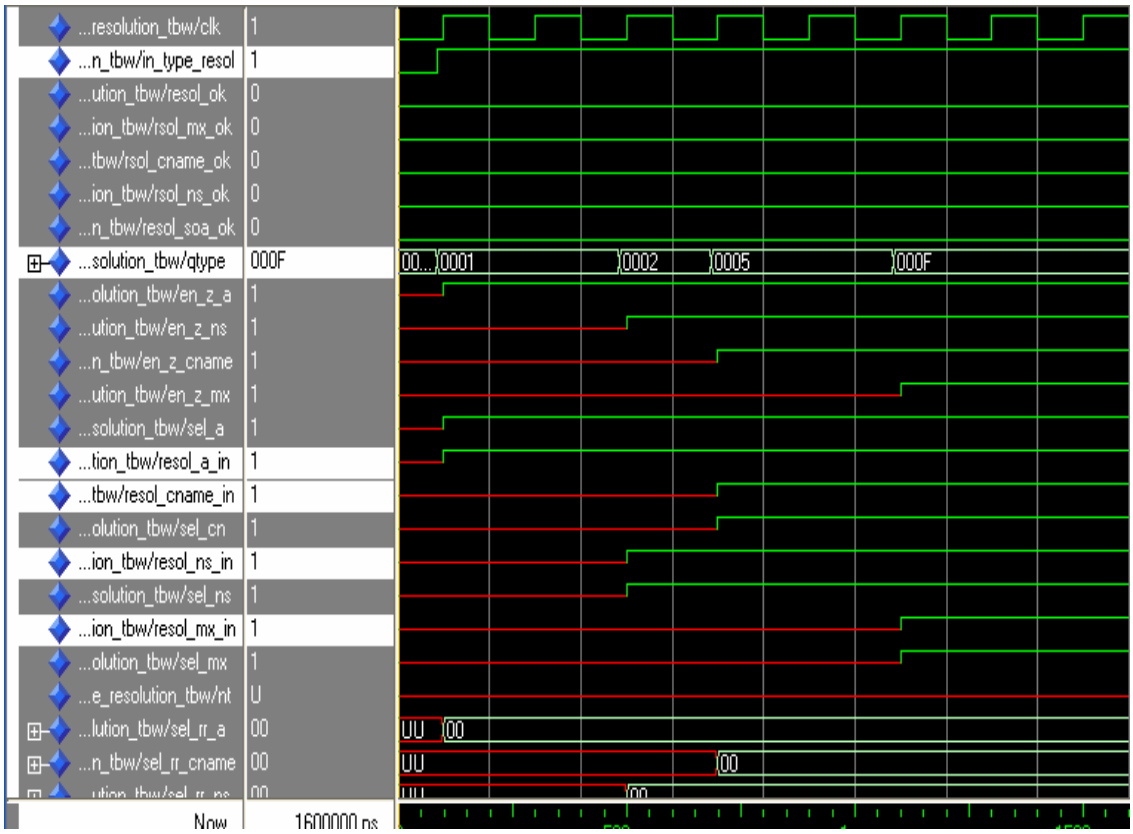




Figures 4.38 : Résultat de la simulation du module *contrôle_resolution*



Figures 4.39 : Résultat de simulation du module *control_type_requete*



Figures 4.40 : Résultat de simulation du module *control_type_resolution*

L'interconnexion des modules composant le sous système résolution, par une description structurelle à conduit au schéma de la figure 4.41.

Ci après la vue externe du bloc *resolution* généré avec l'outil RTL schematic

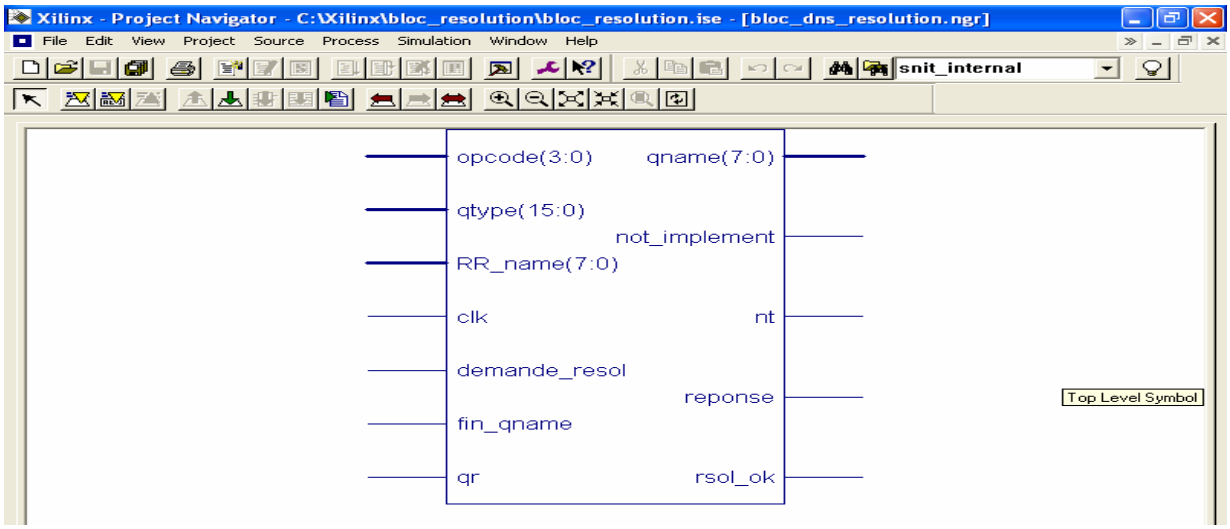


Figure 4.41 : Circuit *resolution* synthétisé

La structure interne du sous système résolution est représentée par la figure suivante

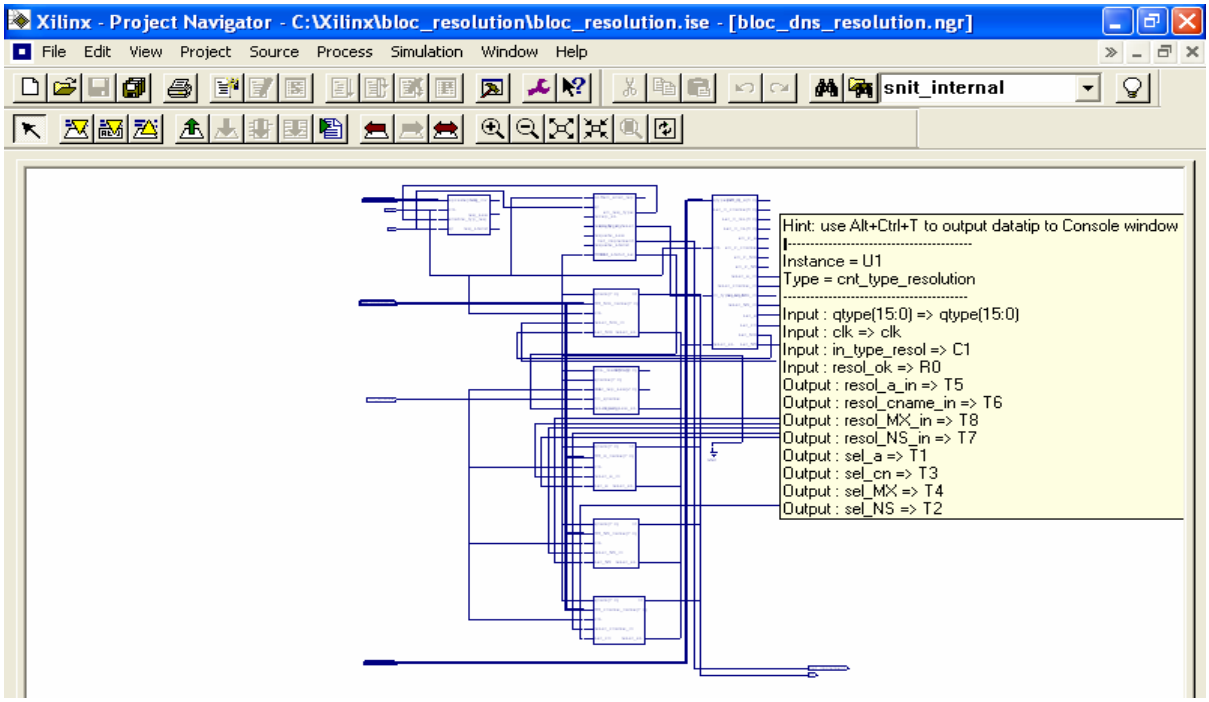


Figure 4.42 : Structure interne du circuit (bloc *resolution*) synthétisé

4. 3.4 Bloc Emission

4. 3.4 .1 Description fonctionnelle

La figure ci après illustre l'algorithme de fonctionnement du bloc *emission.*

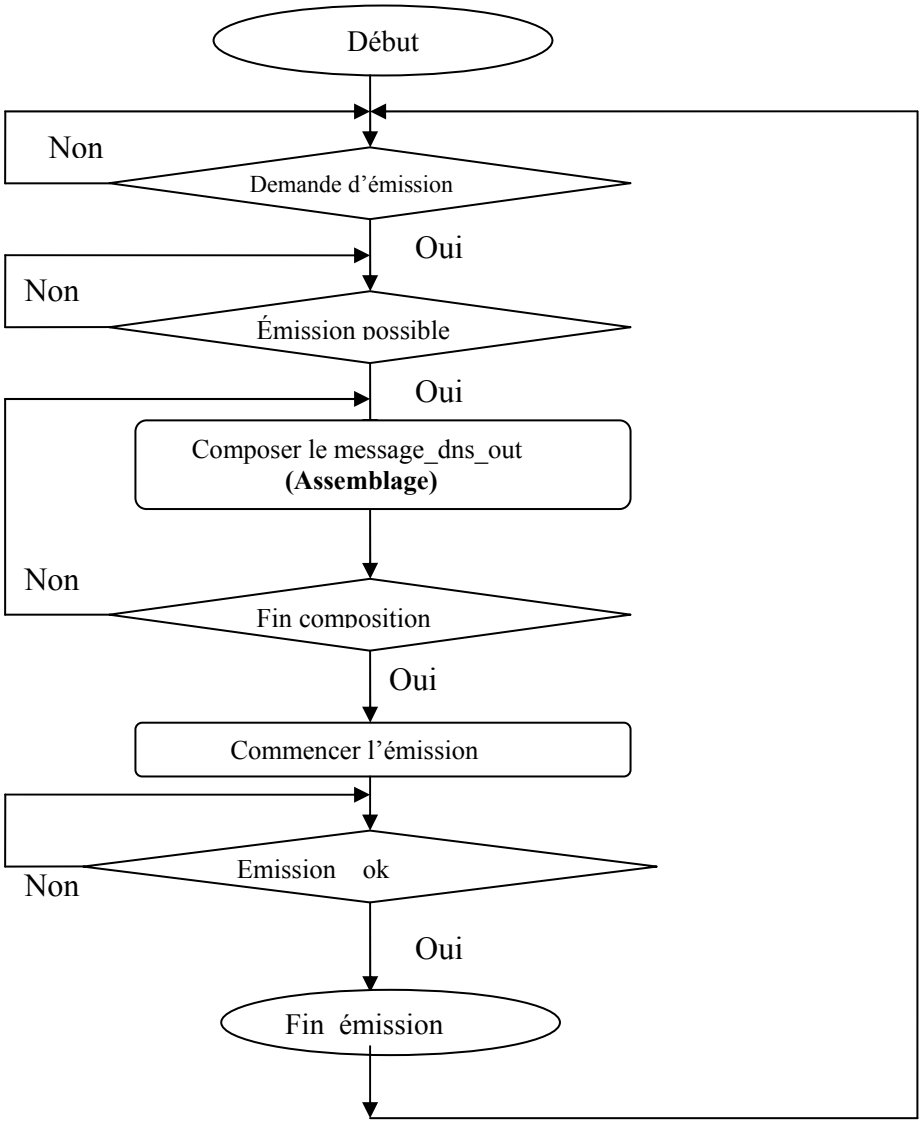


Figure 4.43 : organigramme de l'émission

Ce bloc représente l'interface de sortie de notre système. Sa fonction essentielle est l'inverse de celle du bloc *reception*. En effet, son rôle premier consiste à assembler les champs fournis par le bloc *resolution* et de composer le *message_dns_out* à partir des différents champs constituant la réponse ; puis à transférer le message à transmettre (la réponse) sous forme de successions d'octets en ajoutant l'information pour les champs *Rcode* (4bit) et le bit *AA* .

La chaîne de caractères ainsi construite par une concaténation, sera envoyée au client, via un bus de donnée qui est spécifié sur 8 bits.

Pour se faire, l'interface dialoguera avec la fifo de sortie (*fifo_out*) et le bloc *resolution*.

Quand la résolution est effectuée et le bloc *resolution* désire envoyer une réponse, il met le signal *reponse* à '1'.

On attend alors que le signal d'entrée *reponse* passe à '1' , le module *contrôle_emission* réagissant sur le front montant de ce signal enverra un signal *sinit* valant'0' à la *fifo_out*. Ici le signal *sinit* est le signal d'initialisation de la *fifo_out*. Ce signal reste à '0' tout le temps de l'émission.

On transmet alors octet à octet l'information au moyen du signal *message_dns_out* de la manière suivante :

On transmet alors dans l'ordre :

- L'entête du message réponse constitué de 12 octets.
- La question qui est la même que celle tirée du *message_dns_in* est de longueur variable (259 octets max).
- La réponse qui est le résultat de la résolution de nom, inclut l'information (le RR) pour le nom de machine demandé. Elle est constitué de 254 octet max pour le nom de la machine en plus des champs constituants l'enregistrement de ressource tel qu'il est illustré dans les fenêtres structure montrés dans les section précédentes .

A tout moment, on scrute le signal *empty_2* provenant de la *fifo_out* ci celui-ci passe à '1' alors on stoppe l'émission en mettant le signal *sinit* à 1.

La figure suivante représente la structure interne du bloc *emission*

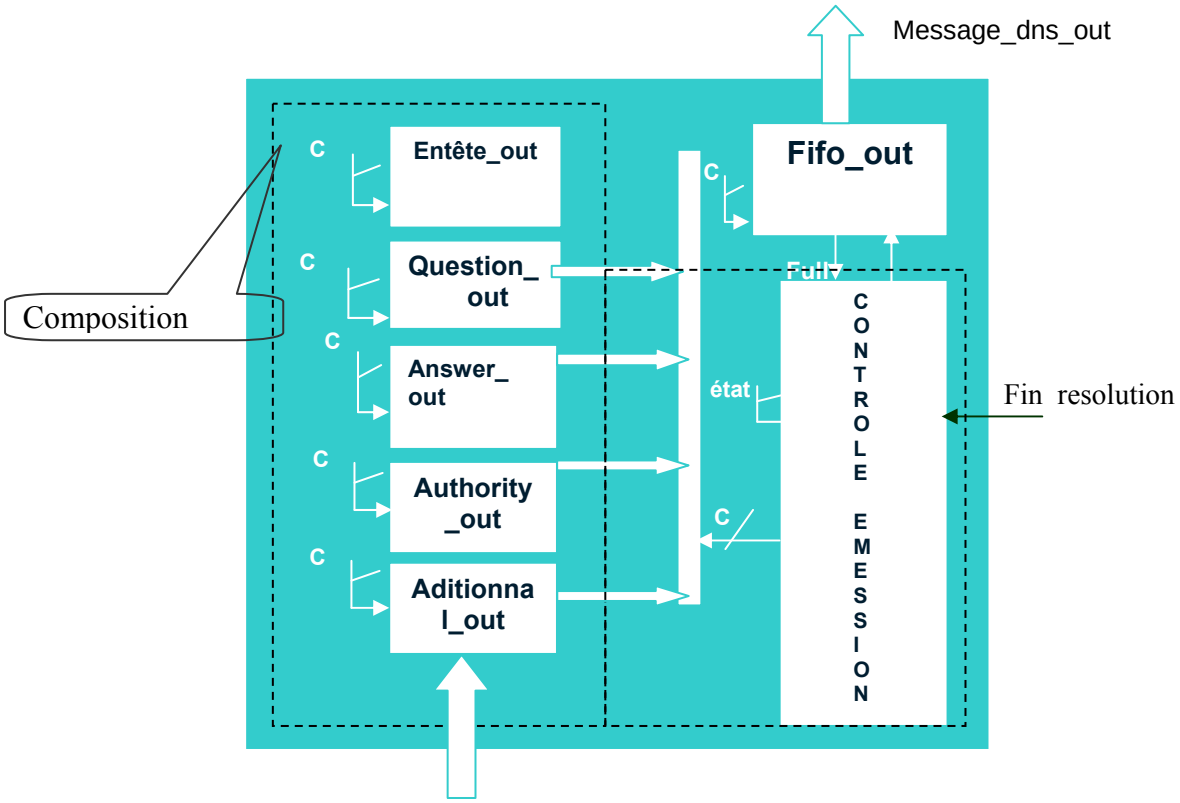


Figure 4.44 : Architecture interne du bloc émission

4.3.4.2 Module fifo_out

4.3.4.2 A Fonctionnement

Fifo_out signifie une fifo en sortie, sa structure et son fonctionnement sont identiques à ceux de la *fifo_in* du bloc *reception*. C'est une fifo circulaire d'une profondeur de 512 mots de 8 bits.

Comme pour le bloc *reception*, ce module sert de tampon pour le bloc *emission* .Le *message_dns_out* (la réponse à la requête) est transmis au client via un bus de 8 bits (octet par octet). La figure 4.45 représente la vue externe de ce module.

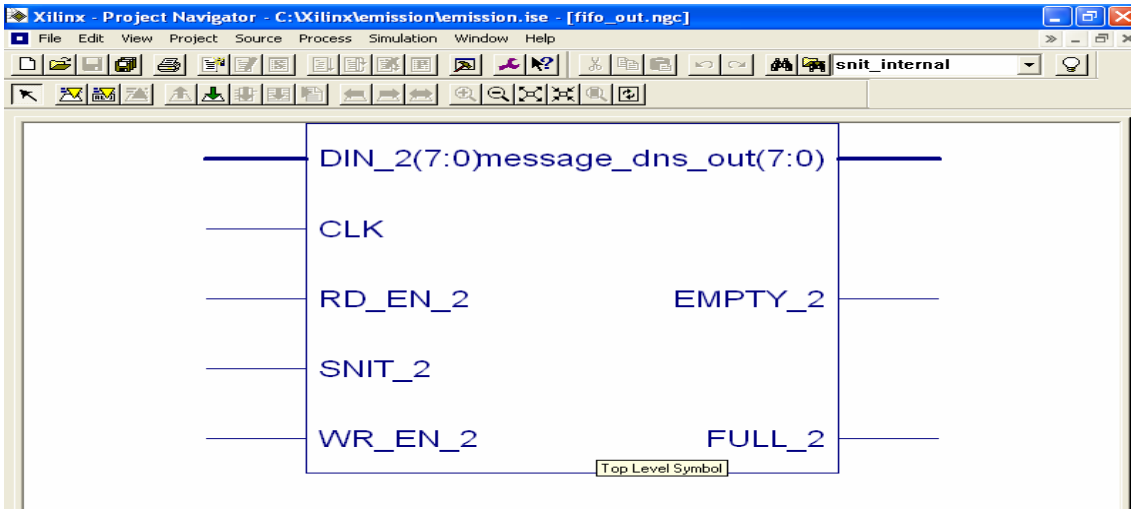


Figure 4.45 : Vue externe du module *fifo_out*

La description des entrées et des sorties de la *fifo_out* sont données dans le tableau 6

Nom	Direction	Largeur	Description
Clk	In	1 bit	Horloge.
Sinit_2	In	1 bit	Remise à zéro, lorsque sinit ='1' la fifo est déclaré vide (cette entrée annule le contenu de la mémoire)
Wr_en_2	In	1 bit	Entrée active sur front montant, elle correspond à une demande d'écriture.
Rd_en_2	In	1 bit	Active sur front montant, elle correspond à une demande de lecture.
DIN_2	In	8 bits	Vecteur de données, il contient les données qui seront stockées dans la fifo après une demande d'écriture.
Message_dns_out	Out	8 bits	Vecteur de données en sortie, il est mis à jour après une demande de lecture.
Empty _2 (vide)	Out	1 bit	Signal d'état est actif (prend la valeur '1') lorsque la FIFO est vide.
Full_2 (pleine)	Out	1 bit	Signal d'état est actif (prend la valeur '1') lorsque la FIFO est pleine.

Tableau 6 : Entrées et Sorties de la *fifo_out*

4.3.4.2 B Simulation

Pour le test de ce module la chaîne de caractère représentant le *message_dns_out* est envoyée dans la *fifo_out* via le module lecture fichier (module de test identique à celui utilisé pour le test du bloc *reception*). L'entrée de ce module est la succession d'octets du *message_dns_out* (une chaîne de caractères de 58 octets).

Le résultat de simulation du module *fifo_out* est donné ci après ,il représente une réponse à une requête de type A.

Les figures suivantes illustrent une partie de la réponse « message_dns_out ».

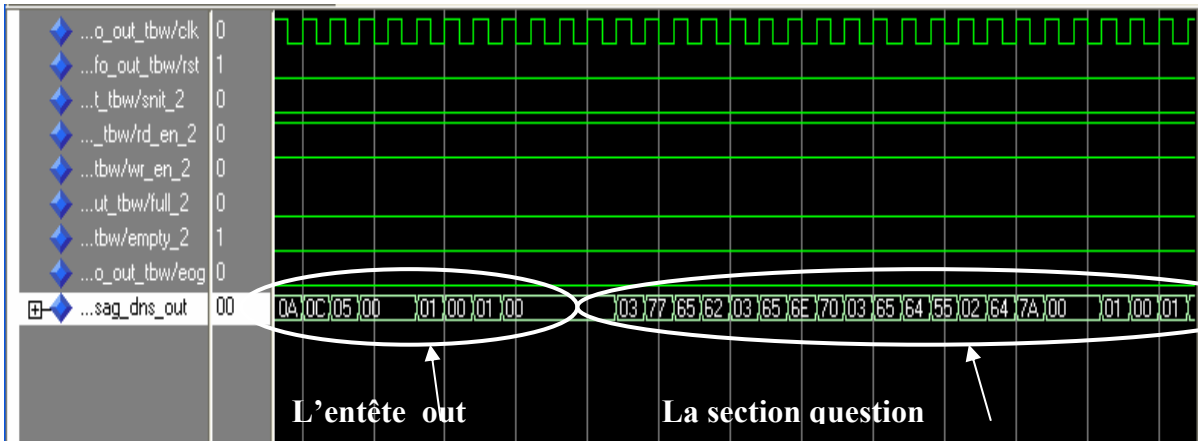


Figure 4.46.a : Résultat de simulation du module *fifo_out*

La section answer (la suite du *message_dns_out*) est donnée par la figure ci après

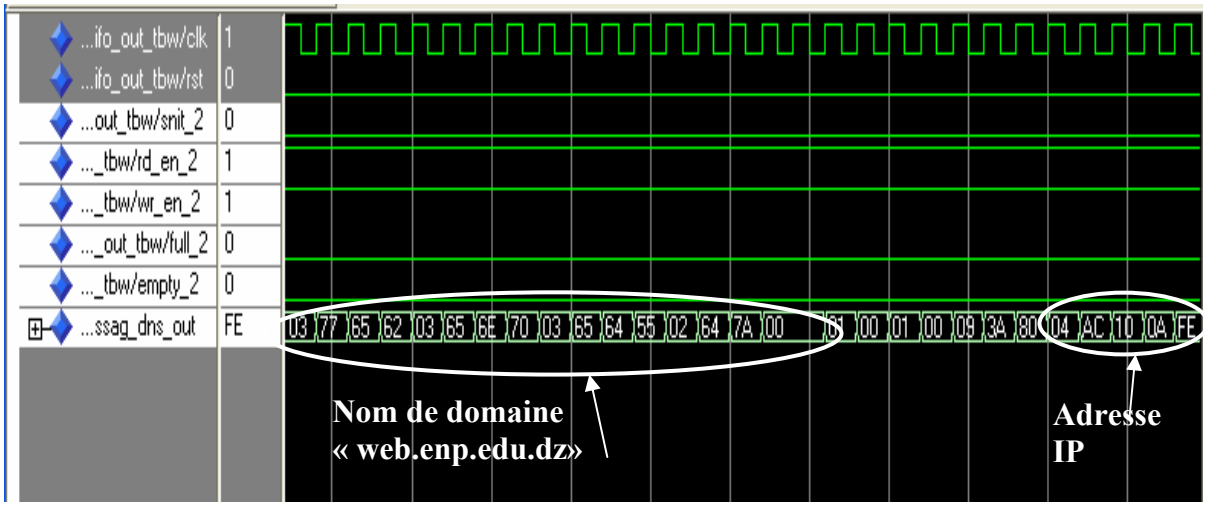


Figure 4.46.b : Résultat de simulation du module *fifo_out*

Remarque : les sections authority et additionnel son vide dans ce cas.

4. 3. 4.3 Sous bloc composition

4.3. 4.3.1 Module entete_out

4.3. 4.3.1 A Fonctionnement

Ce bloc est un circuit de composition des champs que doit contenir l'entete_out. Il communique avec les blocs *resolution* et le module *control_emession*, la sortie de ce module dout_e est envoyée à la *fifo_out.* Il inclue comme pour le module *decomp_pretraitement_ entête* une mémoire de douze mots de 8 bits accessible en lecture et en écriture. Le contrôle de ce module est effectué par les signaux de commande d'écriture et de lecture. Ces commandes sont gérées par le module *contrôle_émission*.

En plus des signaux de lecture et d'écriture, nous avons défini aussi le signal *comp_in* qui correspond à un ordre de composition des champs de l'entête du message en sortie. La particularité pour ce module est l'assignation de valeur correspondante au champ *rcode* selon le résultat de traitement du bloc *resolution*.

La figure 4.47 représente la vue externe de ce module obtenu avec l'outil RTL shematic.

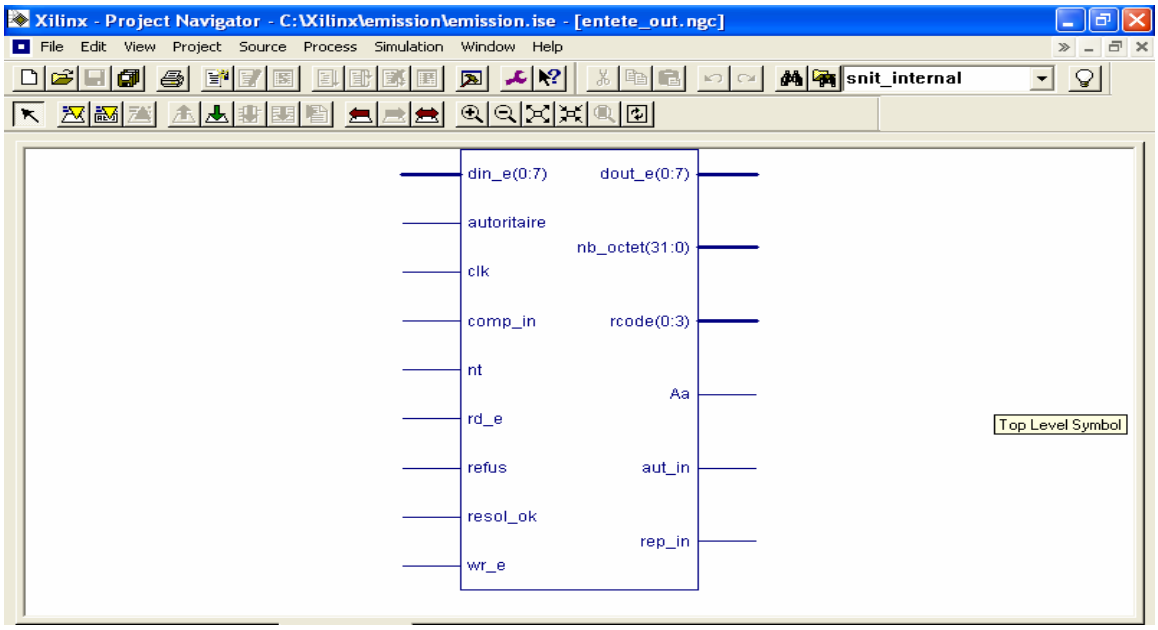


Figure 4.47 : Vue externe du module *entete_out*

Le Tableau 7 récapitule la description des signaux d'entrée/sortie du module *entete_out*

Port	Direction	Largeur	Description
Clk	In	1 bit	Horloge active haute.
comp_in	In	1 bit	Entrée de sélection active sur front montant correspondant aussi à ordre de composition
Wr_e	In	1 bit	Entrée active sur front montant, elle correspond à une demande d'écriture.
rd_e	In	1 bit	Active sur front montant, elle correspond à une demande de lecture.
Din_e	In	8 bits	Ce vecteur est la donnée à insérer dans Ce module après une demande d'écriture
Dout_e	Out	8 bits	Vecteur de données en sortie, représente la sortie retirée de ce module.
autoritaire	In	1 bit	Quand ce bit est à '1' la réponse et d'autorité
Resol_ok	In	I bit	Cette entrée provient du module résolution indiquant que le nom est trouvé si ce bit est à '1 '.
NT	In	1 bit	de même cette entrée provient du module résolution indiquant que le nom **n'est pas trouvé** si ce bit est à 1
Refus	In	1 bit	Ce signal provient du module statut_serveur indiquant que le transfert de zone est autorisé.
Rcode	Out	4 bits	Sa valeur spécifie le type de réponse
Aa	Out	1 bit	Sortie qui prend la valeurs '1'pour indiqué que la réponse et d'autorité (cette sortie est à '0' dans le cas contraire

Tableau 7 : Entrées et Sorties du module *entete_out*

4.3. 4.3.1 *B Simulation*

Le résultat pour le test de ce module est donné ci après. Il représente la partie entete_out du *message_dns_out*. On peut remarquer essentiellement que le champ *rcode* de 4 bits vaut « 0000» quand la valeur du bit « *NT*» est à'0' et celle du signal

resol_ok est égale à '1' ; ce qui signifie que la réponse est trouvée . Ce champ passe à « 0011 » dans le cas contraire.

Aussi le test de simulation montre que le bit « Aa » passe à '1' avec le passage de l'entrée autoritaire à 1 indiquant que la réponse est issue des fichiers de zone disponibles en local.

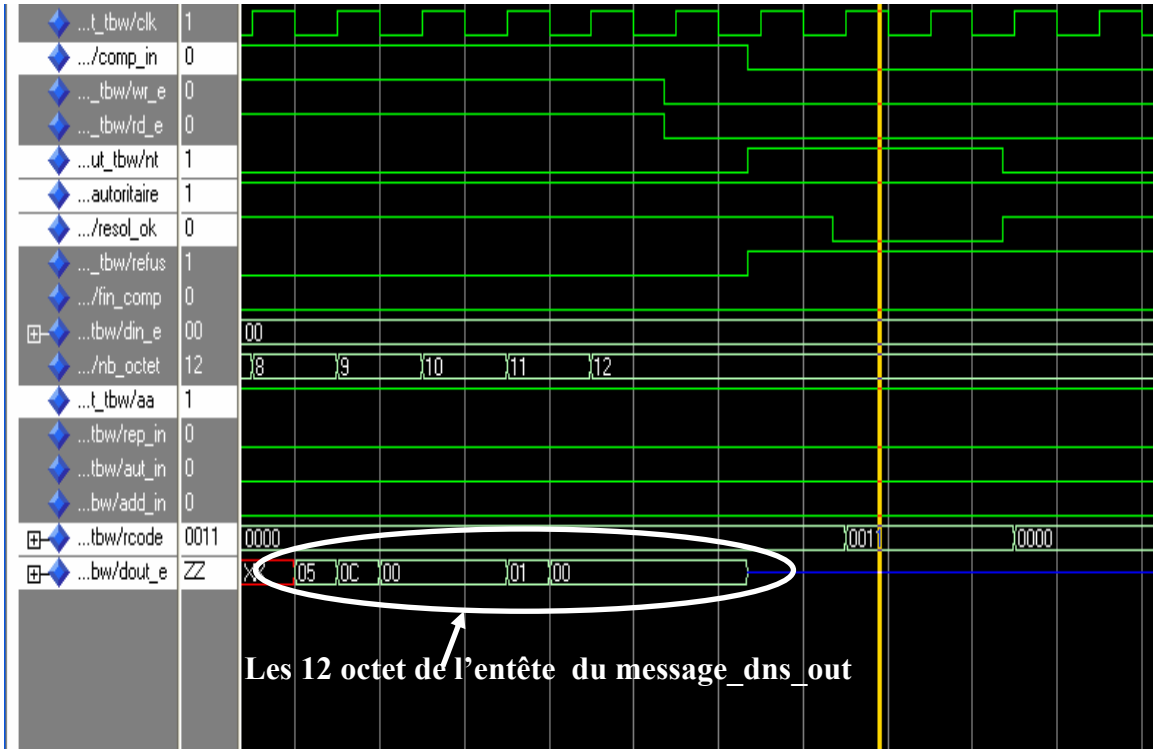


Figure 4.48: Résultat de simulation du module *entete_out*

4.3. 4.3.2 Module question_out

4.3. 4.3.2 A Fonctionnement

Ce module représente la section question. L'organisation interne de ce module est identique à celle du module question du bloc *reception*. Comme illustrée par la figure ci-dessous, la seule différence est au niveau des ports de sortie .Pour question_out, on ne déclarera qu'une seule sortie dou_q. Les sorties qname, qtype, qclass sont supprimées puisque elles servent de vecteur d'états pour le module *control_reception* et sont inutiles dans le cas de l'émission.

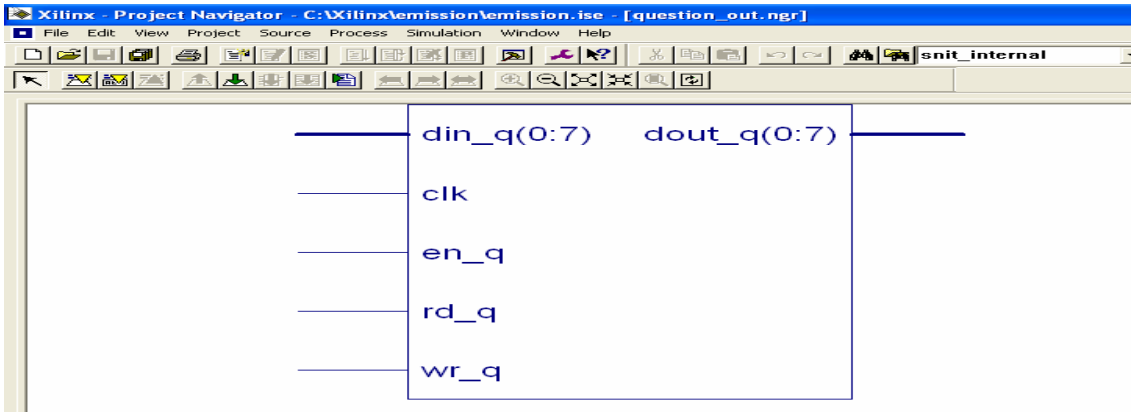


Figure 4.49 : Vue externe du module *question_out*

4.3. 4.3.2 B *Simulation*

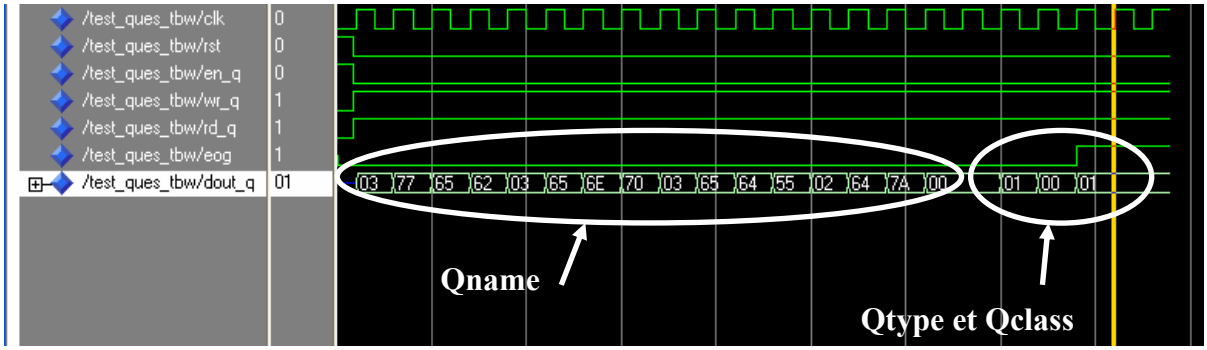


Figure 4.50 : Résultat de simulation du module *question_out*

4.3. 4.3.3 Module answer_out

Notons que la structure interne des modules answer_out, additional_out, authority_out, du bloc *emission* est identique à ceux du bloc *reception*. De ce fait, nous ne donnerons dans ce qui suit que les résultats de simulation pour le module *answer_out* qui contiendra le RR réponse de type A.

4.3. 4.3.3.1 *Simulation*

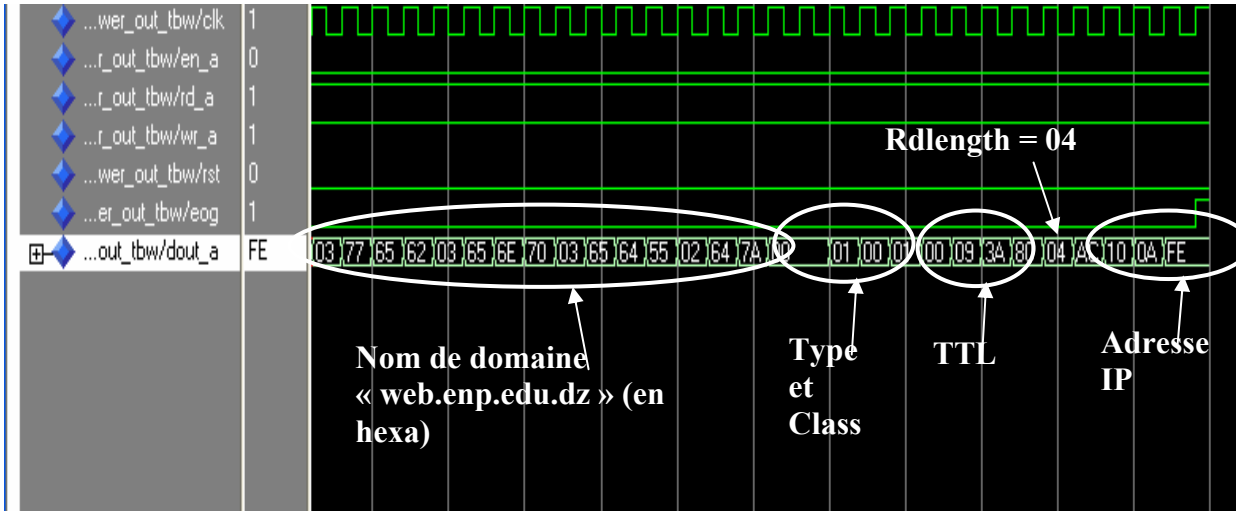


Figure 4.51 : Résultat de simulation du module *ansewr_out*

4.3. 4.4 Module control_emission

4.3. 4.4.1 A Fonctionnement

La vue externe du module *control_emission* est donnée ci-dessous. Sa tâche est plus simple que celle du *control_reception*. En effet, le prétraitement au niveau émission est moins complexe qu'en réception puisque le module *entete_out* (décrit précédemment) effectue une partie du contrôle en traitant quelques signaux en provenance du bloc *resolution*. Ces signaux sont les bits *resol_ok, NT*, et *refus* facilitant ainsi la tâche au module *control_emission*. Le rôle premier de ce module consiste à gérer la communication entre les sept modules composant le bloc *emission*.

Le module *control_emission* attend que le signal d'entrée *reponse* passe à '1' pour envoyer un signal *sinit* valant '0' à la *fifo_out* après avoir scruter le signal d'état *full_2* (pour tester si la *fifo_out* n'est pas pleine).

Comme l'envoie du *message_dns_out* se fera par sections, la tâche principale de ce module est la génération de l'ensemble des signaux de commande allant vers les modules du bloc *emission* afin d'assurer une transmission de la chaîne d'octets de la réponse dans l'ordre donné dans la section 4. 3.4.1.

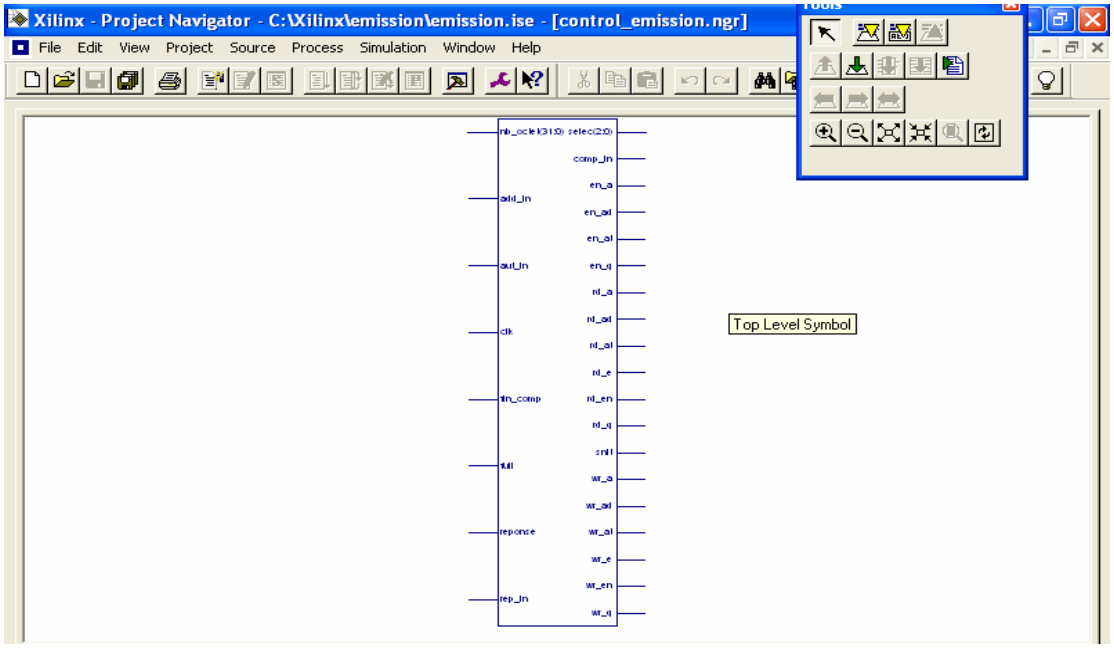


Figure 4.52 : Vue externe du module *control_emission*

4.3. 4.4 *B Simulation*

Voici les résultats de simulation pour ce module**.**

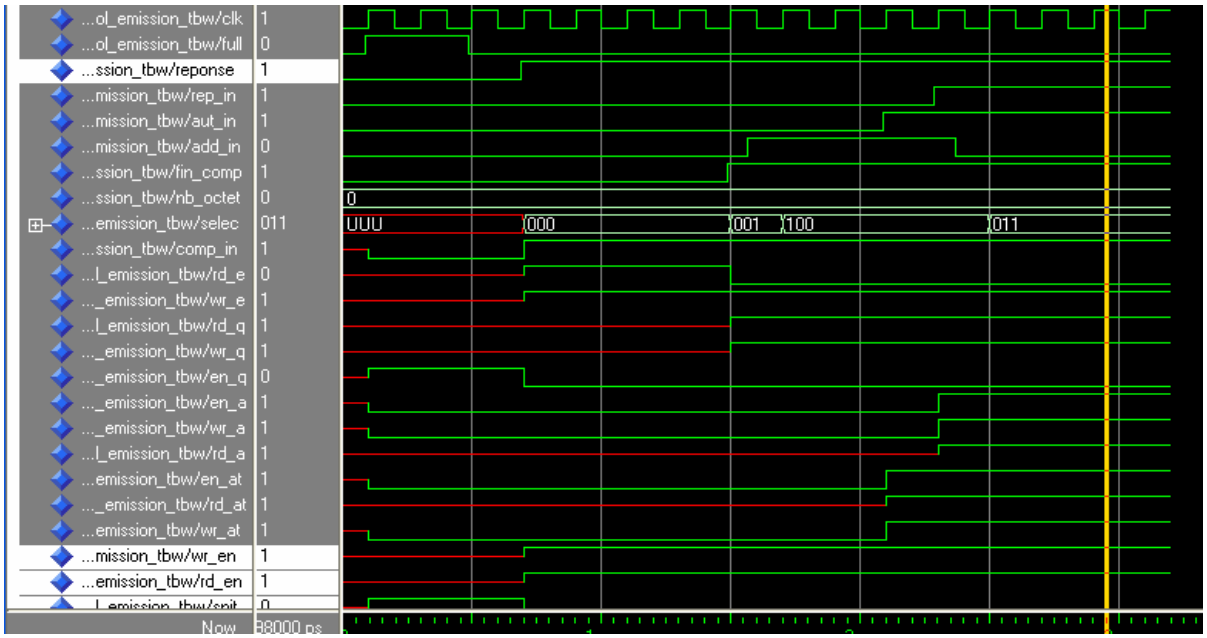


Figure 4.53.a : Résultat de simulation du module *control_emission*

Suite

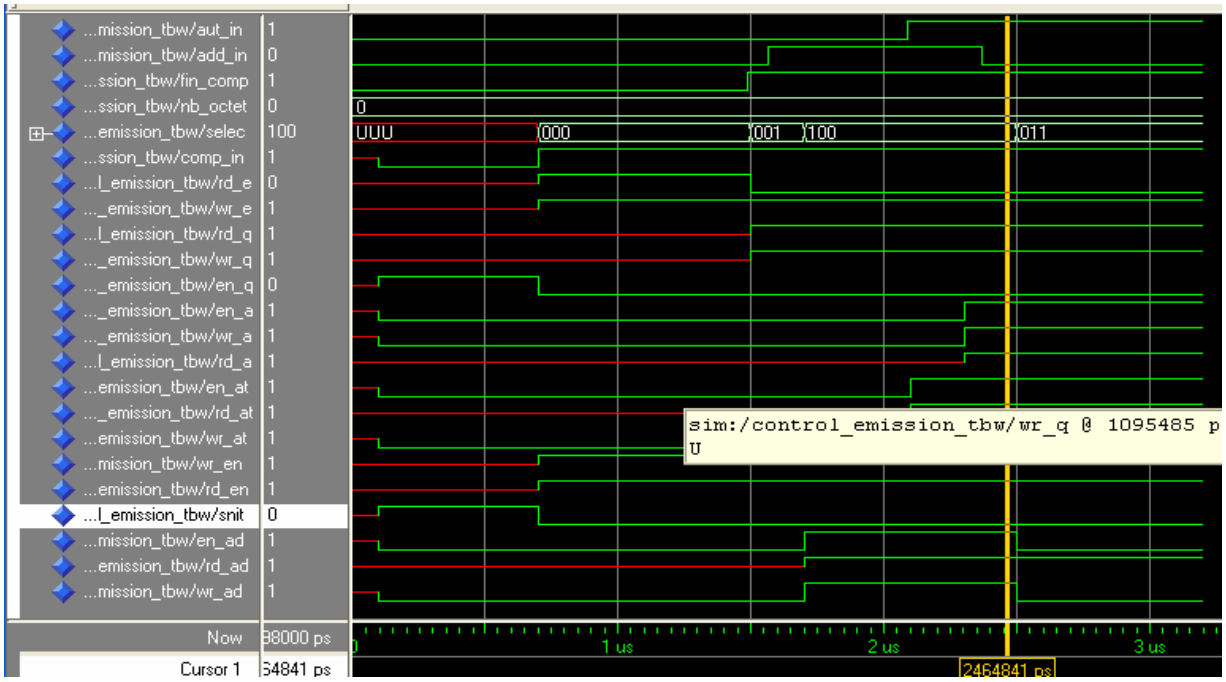


Figure 4.53.b : Résultat de simulation du module *control_emission*

Vu externe du bloc *emission* généré avec l'outil RTL schematic

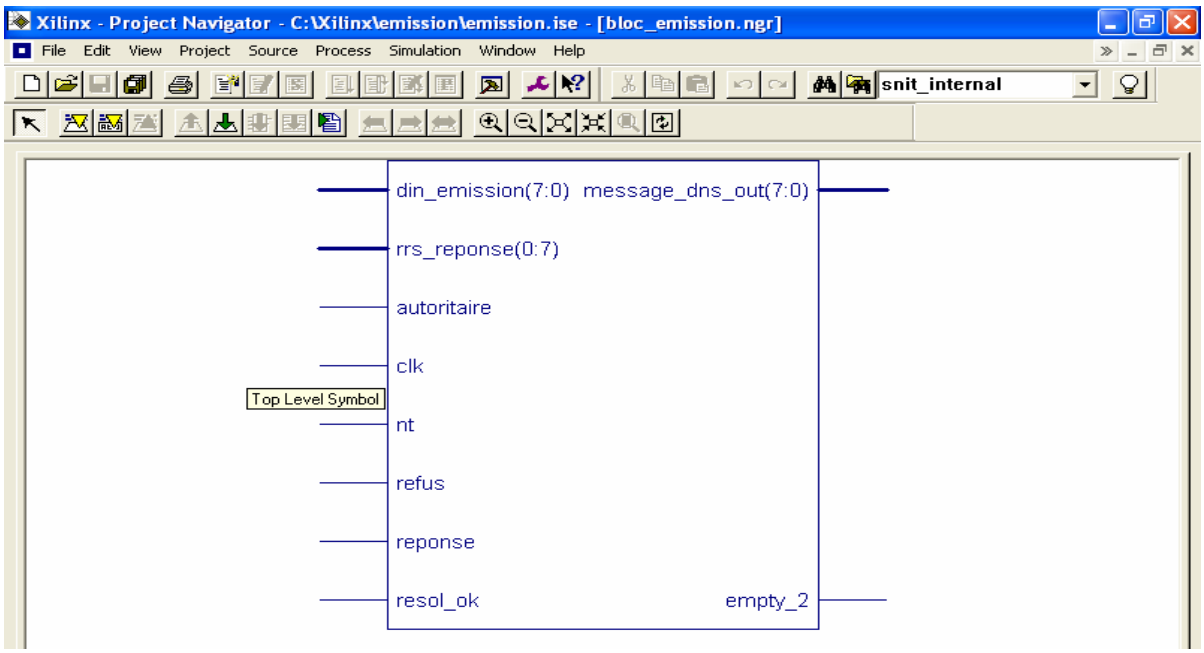


Figure 4.54 : Circuit *emission* synthétisé

La structure interne du sous système émission est donnée par la figure suivante

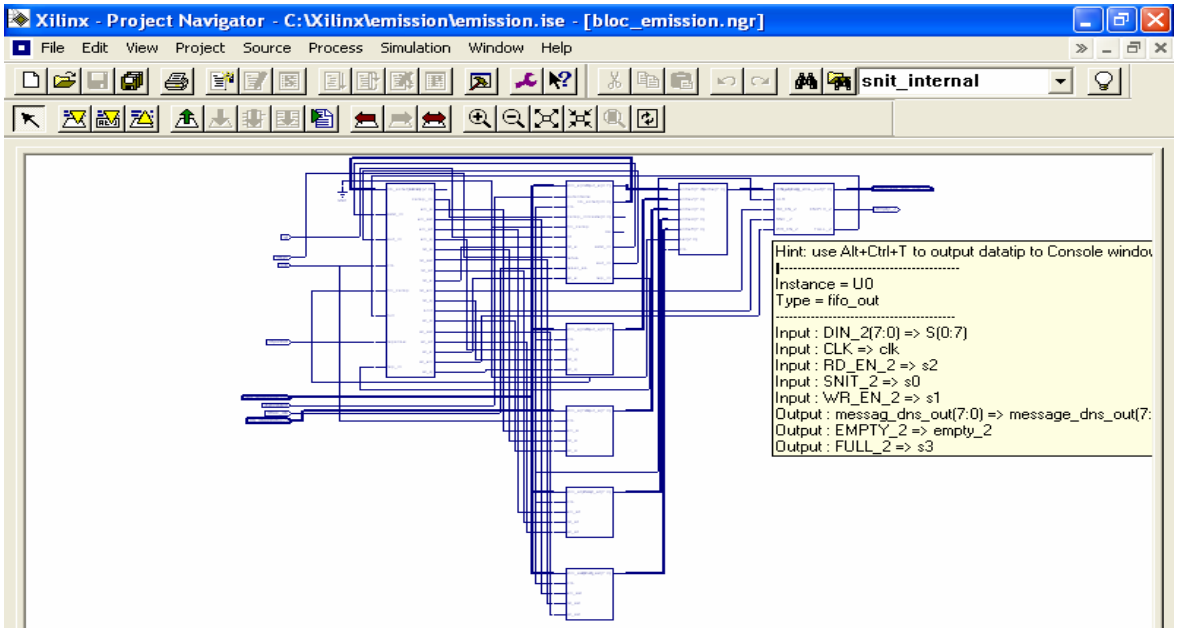


Figure 4.55 : Structure interne du circuit (bloc *emission*) synthétisé

4.4 Implémentation physique

Les figures ci après obtenues avec l'outil graphique le «FPGA Editor» représentent, la configuration du circuit FPGA , après partitionnement, placement et routage tenant compte du circuit FPGA cible (Spartan 3)[14] pour les blocs *reception* et *emission* respectivement figure 4.56 et figure 4.57.

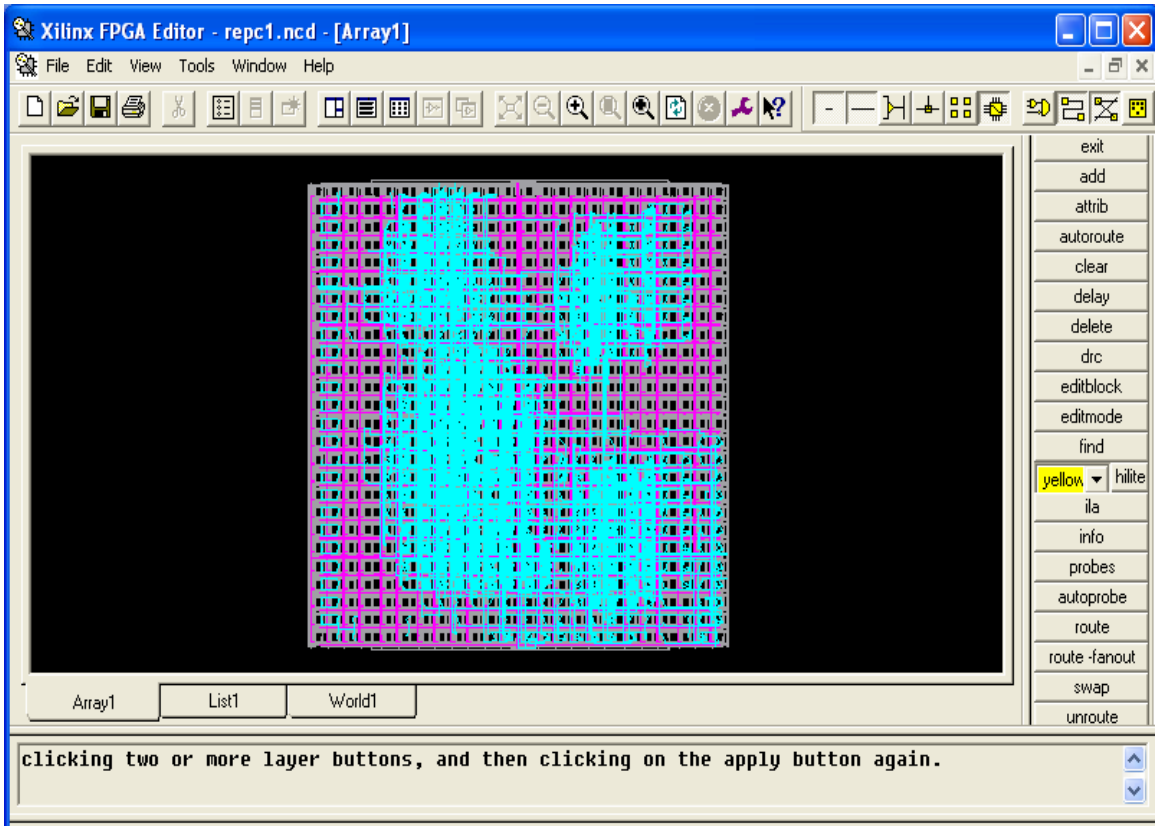


Figure 4.56 : Configurations du circuit FPGA Spartan 3 (*bloc reception*)

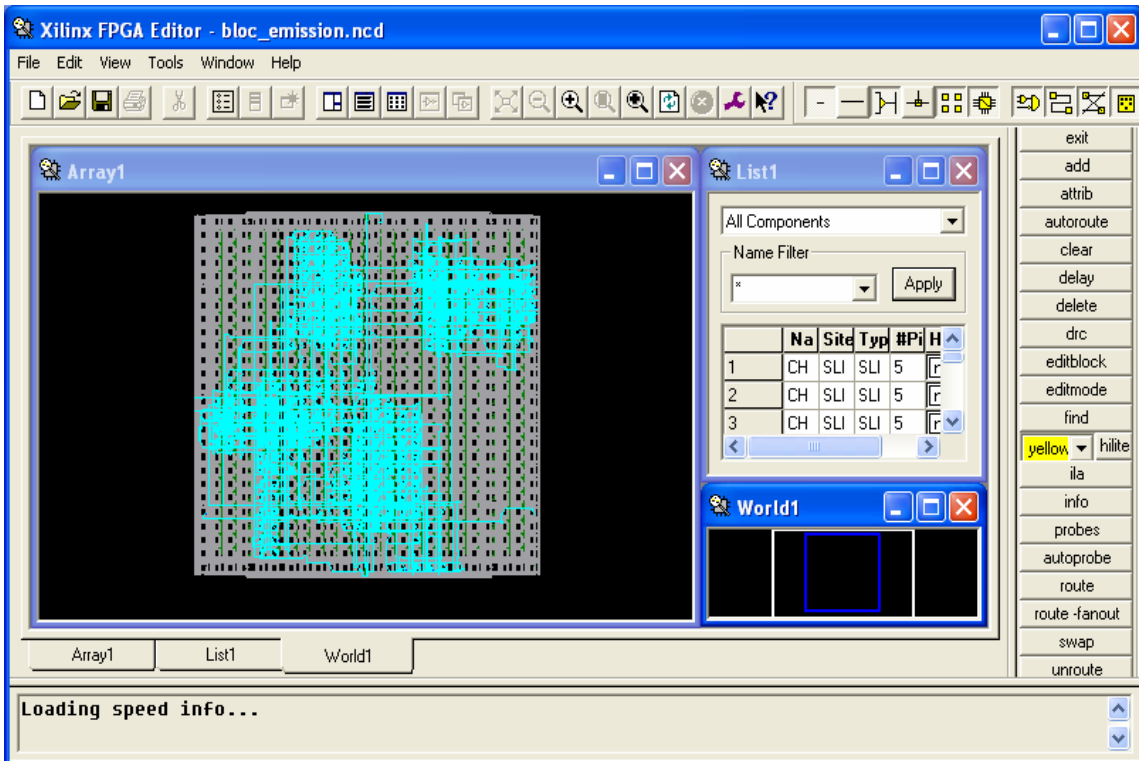


Figure 4.57: Configurations du circuit FPGA spartan 3 (*bloc emission*)

Les rapports ci-dessous constituent un inventaire des ressources utilisées par les blocs *reception* et *emission* après implémentation sur la cible FPGA. Les ressources utilisées représentent environ 35% des ressources globales pour le bloc *reception* et 30% pour le bloc *emission*. Notons que les ressources de la spartan3 sont insuffisantes pour l'implémentation du bloc *resolution.*

==========
Final Report * *reception
==========
Device utilization summary:
- -
Selected Device: 3s200ft256-4
Number of Slices: 682 out of 1920 35%
Number of Slice Flip Flops: 524 out of 3840 13%
Number of 4 input LUTs: 1035 out of 3840 26%
Number of bonded IOBs: 116 out of 173 67%
Number of GCLKs: 1 out of 8 12%

Timing Summary:

Speed Grade: -4
Minimum period: 25.483ns (Maximum Frequency: 39.241MHz)
Minimum input arrival time before clock: 4.120ns
Maximum output required time after clock: 7.241ns
Maximum combinational path delay: No path found
Timing Detail: All values displayed in nanoseconds (ns)
==========
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 25.483ns (frequency: 39.241MHz)
Total number of paths / destination ports: 53967739/ 821==========

Figure 4.58 : Rapport de synthèse pour le bloc *reception*

==========
*** Final Report * *emission***
==========
Device utilization summary:

Selected Device : 3s200ft256-4
Number of Slices: 592 out of 1920 30%
Number of Slice Flip Flops: 428 out of 3840 11%
Number of 4 input LUTs: 758 out of 3840 19%
Number of bonded IOBs: 31 out of 173 17%
Number of GCLKs: 3 out of 8 37%

Timing Summary:

Clock Information:
Speed Grade: -4
Minimum period: 25.483ns (Maximum Frequency: 39.241MHz)
Minimum input arrival time before clock: 4.202ns
Maximum output required time after clock: 7.281ns
==========

Figure 4.59 : Rapport de synthèse pour le *bloc emission*

4.5 Conclusion

L'implémentation que nous avons conçue et réalisée sous forme de bloc IP est constituée d'un ensemble de blocs fonctionnels communicants décrits à l'aide du langage de description VHDL; la cible technologique choisie étant un FPGA. L'architecture établie a tenu compte des définitions des règles de normalisation édictées par la communauté Internet.

Conclusion et perspectives

Notre développement constitue une contribution à la réalisation d'un serveur DNS sous forme matérielle. A travers ce développement, nous avons mis en valeur notre approche de conception d'un serveur autoritaire matériel. Les résultats graphiques nous ont emmené à valider notre conception.
Au cours de ce travail, nous avons établi un premier modèle architectural caractérisé par sa modularité pour une meilleure performance et une plus grande sécurité. L'implémentation obtenue permet un isolement physique beaucoup plus complet. Ainsi, le service DNS s'exécutant sur un circuit est isolé des autres applications tant au point de vue ressources internes que du réseau.

L'intérêt de notre conception est donc double; garantir l'intégrité du service et donc une bien meilleure continuité du service d'une part et d'autre part obtenir de meilleures performances de fonctionnement locales réduisant ainsi les latences liées au système de traitement. Il est clair aussi que cette forme d'implémentation ne pourra que mieux garantir l'intégrité des données gérées localement vis à vis d'événements exogènes.

Lors de la phase de validation, trois IP core ont été créés ; à savoir les IP core *reception*, *emission*, *resolution*. La carte cible pour la validation est une carte à base de circuit FPGA spartan3. Nous étions contraints d'opter pour une implémentation et validation des IP core séparément car la carte dont nous disposions était limitée en ressources par rapport à notre conception.
A défaut d'une implémentation d'un serveur DNS autonome pour l'évaluation effective des performance de notre architecture, nous précisons que le modèle architectural que nous proposons se situe dans la couche application du modèle

TCP/IP et que le travail que nous avons présenté n'a pas d'antécédents connus et surtout que notre architecture présente la brique de base pour la réalisation d'un serveur DNS autonome , facilitant ainsi la tâche pour d'éventuelle(s) exploitation(s) ou amélioration de notre travail .

Il conviendra aussi de noter que l'architecture proposée présente une solution intéressante parce qu'elle peut être un modèle de référence pour tout système possédant la même structure que le DNS et nécessitant une sécurisation.

Plusieurs perspectives de développement restent à explorer. Nous préconisons, comme première perspective, l'intégration des fonctionnalités des couches inférieures du modèle OSI avec la même approche d'implémentation matérielle en utilisant le langage VHDL.

Bibliographie

[1] P.Mockapetris, RFC 1034 “Domain names-concept and facilities”, November 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt.

[2] P.Mockapetris, RFC 1035 “Domain names-implementation and specification”, ISI, November1987. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[3] Jack Tckett, David Gunter & Lance Brown, LINUX, S&SM, 20 Novembre 1995.

[4] Internet - Noms de domaines .URL http://www.commentcamarche.net/

[5] Bind 9 administrator reference Manuel .Copyright 2000, 2001 by Internet Software Consortium [Online]. Available : www.bind9.net/manuels

[6] Bertrand Léonard, “Sécurisation du DNS : les extensions DNSsec “, Octobre2003,
AFNIC//projet IDsA . URL http://www.idsa.prd.fr

[7] URL : www.ethereal.com

[8] URL : www.commentcamarche.net

[9] Jürgen Pfleger , “ DNSSEC Resolver Algorithm “ (Master Thesis) Fachhochschulstudiengänge Technik Austria ,04/2003.

[10] Ammar Attoui , " Architecture des systèmes sur puce : Processeurs synthétisables, CAO VLSI, norme VCI, environnement ISE et Quartus " , Paris : Ellipses,2005.

[11] R. Airiau, J.M. Bergé, V. olive, [et al.], VHDL : Langage, Modélisation, Synthèse /.-2 éd. rev. et augm . Lausanne : Pressses Polytechniques et Universitaires Romandes,1998 .

[12] Hakim Saheb et Michel Dang , : " An architecture synthesis of low-level communication circuits: ", ASIC Conference and Exhibit.. Proceedings., Sixth Annual IEEE International ,27 Sep-1 Oct 1993 IEEE .

[13] URL : http://www.xilinx.com/bvdocs/appnotes/xapp473.pdf

[14] Spartan-3Starter Kit Board User Guide (Manuel) April 26, 2004.