REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud MAMMERI de Tizi-Ouzou

Faculté de Génie Electrique et d'Informatique

Département d'informatique

MEMOIRE

DE FIN D'ETUDE

En vue de l'obtention d'un diplôme de Master Académique en Informatique.

Option : Réseaux, Mobilité et Systèmes Embarqués.

Mémoires TCAM pour caches d'un routeur NDN Sur circuit FPGA (Altera DE2)

Réalisé par :

- *Melle HADDADJ Lylia.*
- *Melle FERRAH Zina.*

*Mémoire soutenu publiquement le **25/09/2018**. Devant le jury composé de :*

Présidente : *Mme BELKADI*

Examinatrice : *Mme HADAOUI*

Encadreur : *Mr M.DAOUI*

Promotion 2017-2018

Dédicaces

Je dédie ce modeste travail à :

Mes très chers parents qui sont la source de ma réussite.

A ma chère sœur.

A mon cher frère.

A ma famille.

A mes amis.

-Lylia-

Je dédie ce modeste travail à :

Mes très chers parents qui sont la source de ma réussite.

A mes chères sœurs.

A mes chers frères.

A mon cher mari.

A ma famille.

A mes amis.

-Zina-

Table des matières

Table des matières

Table des matières

Table des matières

Introduction Générale

Introduction générale

L'Internet a révolutionné le monde des ordinateurs et des communications comme rien d'autre auparavant. L'invention du télégraphe, du téléphone, de la radio et de l'ordinateur a ouvert la voie à cette intégration sans précédent de capacités. L'Internet est à la fois une capacité de diffusion dans le monde entier, un mécanisme de distribution de l'information et d'interaction entre les individus à travers leurs ordinateurs, peu importe l'emplacement géographique. L'information est transmise par Internet grâce à un ensemble standardisé de protocoles de transfert de données. L'Internet a beaucoup changé au cours des deux décennies qui ont suivi sa naissance. Il fut créé à l'époque du temps partagé, mais a survécu jusque dans l'ère des ordinateurs personnels, de l'informatique client-serveur et poste-à-poste. Il a été conçu avant l'apparition des réseaux locaux, mais a accueilli cette nouvelle technologie de réseau. Ce réseau mondial continuera donc à changer et à évoluer à la vitesse de l'industrie informatique s'il doit demeurer pertinent. Il est en train de changer pour fournir de nouveaux services tels que le transport en temps réel, afin de soutenir, par exemple, les flux audio et vidéo. La question la plus pressante pour l'avenir d'Internet n'est pas comment la technologie va changer, mais comment le processus de changement et d'évolution lui-même sera géré.

Avec cette évolution, chaque jour de nouveaux objets se connectent à l'internet (L'internet des objets). Ces objets connectés envoient continuellement des informations concernant leur environnement. Ces données collectées forment une grande base de données. Tous ces objets ont besoins d'une adresse IP afin de pouvoir se connecter, donc ce qui engendre le manque d'adressage malgré la transition vers IPv6 (espace d'adressage sur 128 bits). Vu l'importance de la donnée et non pas son fournisseur, l'adresse IP d'un périphérique réseau (ordinateur, routeur, serveur...) n'est plus intéressante. L'internet du futur vise à mettre en valeur et adresser la donnée elle-même, alors une nouvelle architecture est née connue sous le nom de NDN[1].

Lors de déploiement de cette nouvelle architecture au niveau des périphériques réseau tel les routeurs, des problèmes sont apparus concernant la grande quantité de données produites dans ces réseaux. Ainsi il est nécessaire de les atteindre plus rapidement que possible. Cette architecture permet aux routeurs de sauvegarder les données reçues dans leur cache afin de les garder plus proche de l'utilisateur. Cette politique engendre le stockage de grandes quantités de données dans les caches ce qui rend le processus de recherche difficile et long.

[1] **NDN : N**amed **D**ata **Ne**tworking.

Introduction générale

Face aux difficultés rencontrées au niveau des caches des routeurs, Nous nous sommes intéressés à l'amélioration de ces mémoires, car les caches des routeurs à base de mémoire **SRAM**[2] engendre un temps d'accès énorme. Pour cela nous proposons de remplacer les SRAM par des mémoires **TCAM**[3] qui permettent d'accélérer le processus de recherche.

Ce mémoire est organisé en quatre chapitres :

- Le chapitre 1 intitulé, **L'architecture NDN**, est consacré à la présentation de l'architecture NDN et ses principes.
- Le chapitre 2 intitulé, **Les mémoires TCAM**, est consacré à la présentation des mémoires TCAM et de ses différents types, ainsi ses avantages et son utilisation.
- Le chapitre 3 consacré à la **conception**, décrit en détails l'architecture proposée pour un routeur NDN et le système à implémenter.
- Le chapitre 4 consacré à la **réalisation**, présente l'implémentation matérielle sur circuit FPGA de l'architecture décrite en conception. On présente ainsi le prototype, les simulations et les résultats obtenus.

[2] **SRAM : S**tatic **R**andom **A**ccess **M**emory.

[3] **TCAM : T**ernary **C**ontent **A**ddressable **M**emory.

Chapitre 1 : L'architecture NDN

Chapitre 1 : L'architecture NDN

1.1. Introduction à l'Internet IP et les limitations

Internet a été créé dans les années 70s pour échanger des informations ou communiquer. Les utilisations d'Internet étaient relativement simples, comme rechercher des informations sur des sites web, messagerie instantanée, FTP, etc. Pour ces usages, les communications dans les réseaux sont réalisées selon un modèle de bout-en-bout, en établissant des tunnels de communications, depuis un point de terminal (un utilisateur) vers un autre point. Ce modèle, en vigueur pour les protocoles TCP/IP, était parfaitement adapté à ces usages.

Mais au fur et à mesure, Internet est devenu de plus en plus populaire et de plus en plus utilisé, ce qui a conduit à une évolution d'Internet et des technologies, les réseaux étant plus complexe qu'avant. Les terminaux ne sont plus simplement des ordinateurs bureautiques ou portables, mais sont maintenant des smart phones, des tablettes, des terminaux de jeux vidéo, etc. Les infrastructures réseaux ne se limitent plus à xDSL[4]ou à la fibre optique : les réseaux WiFi, mobile 3G/4G, WSN[5]et satellites sont maintenant largement déployés. La structure de TCP/IP et la couche de transport devient inefficace pour gérer cette pluralité d'environnement. Plusieurs couches et services ont été ajoutés afin de s'adapter à ces besoins. Par exemple NAT et IPv6 ont été introduits pour résoudre le problème de manque d'adresse, La couche sécurité est ajoutée pour protéger les données des utilisateurs. De même pour d'autres fonctionnalités comme le multicast, la mobilité, la communication sans fil. Avec l'apparition de l'internet des objets (**IOT**), le volume du trafic circulant sur le réseau est désormais énorme.

L'effet le plus important de cette progression est le changement des usages d'Internet. Au lieu de se connecter à des serveurs identifiés pour rechercher une information, les utilisateurs souhaitent maintenant récupérer les informations, sans se soucier de l'entité qui leur fournira. Or le modèle original d'Internet de type bout-en bout ou client/serveur n'est pas efficace pour de tels services de distributions de contenus.

1.2. ICN (Information-Centric Networking)

Pour remédier aux inconvénients mentionnés ci-dessus, une nouvelle solution d'Internet, mieux adaptée aux usages Internet actuels a émergé. C'est un nouveau paradigme réseau, dénommé Information-Centric Networking. Les réseaux ICN proposent de changer l'Internet,

---

[4] **DSL :** ***Digital subscriber line*** l'ensemble des techniques mises en place pour un transport numérique de l'information sur une ligne téléphonique

[5] **Wsn :** ***Wireless Sensor Network***, réseau de capteurs capables de recueillir et de transmettre des données d'une manière autonome.

qui est actuellement basé sur les localisations des serveurs avec des adresses bien définies, vers une architecture basés sur le nom des objets, avec les fonctionnalités mentionnées précédemment nativement intégrées. Dans un réseau ICN, les éléments de base du réseau sont les contenus, identifiés avec un nommage précis, plutôt que les machines (e.g. des serveurs, routeurs) identifiés avec leurs adresses IP.

Le réseau ICN est un modèle basé sur le récepteur (receiver-driven). C'est-à-dire qu'un utilisateur exprime seulement son intérêt sur des contenus au réseau. Ensuite c'est le réseau qui a pour charge de trouver les bons contenus et les meilleures sources pour ces contenus, en se basant sur leurs noms. Dans le cadre de ce mémoire, on s'intéresse à un type d'ICN qui est l'architecture **NDN**.

1.3. L'architecture NDN

1.3.1. Une nouvelle architecture d'internet

L'architecture de sablier d'Internet d'aujourd'hui est centrée sur la couche de réseau (IP) qui met en œuvre les fonctionnalités minimales nécessaires à l'inter-connectivité globale. Cependant, IP a été conçu pour créer un réseau de communication, où les paquets ne nommaient que les terminaux (**endpoints**) tel les routeurs, ordinateurs.

La croissance du commerce électronique, des médias numériques, des réseaux sociaux et des applications pour smartphones a conduit à une utilisation dominante d'Internet en tant que réseau de distribution. Ces derniers sont plus généraux que les réseaux de communication, ce qui rend la résolution de problèmes de distribution via un protocole tel IP très complexe.

Le projet NDN a proposé une évolution de l'architecture IP qui généralise le rôle de la communication, de sorte que les paquets peuvent nommer des objets au lieu des terminaux (**endpoints**).

Plus particulièrement, NDN modifie la sémantique du service réseau qui se restreint de délivrer le paquet à une adresse de destination particulière pour récupérer des données, et l'élargie en une sémantique qui identifie les données par un nom spécifique au lieu d'une adresse.

1.3.2. Présentation du projet NDN [1]

L'architecture **NDN** a été initialement proposée via le projet **CCN**[6] et par Van Jacobson en 2009. Ce projet a par la suite été séparé en deux autres : le projet **CCN**, géré par le **Palo Alto Research Center** (**PARC**), et le projet **NDN** de la **NSF** (**National Science Foundation)**, géré par **UCLA**[7]. Ces deux projets ont la même architecture puisqu'ils sont basés sur l'architecture **ICN**. Nous décrivons ci-dessous l'architecture NDN, ainsi que ses composants et opérations de base.

1.3.3. Aperçu sur l'architecture NDN

Pour les réseaux orientés contenu, tel que NDN, les données sont la base des échanges. (Voir la **Figure 1.1**). Chaque contenu (ou donnée) est identifié par un nom (ou préfixe) qui a une structure hiérarchique. Le nom dans un paquet NDN peut nommer n'importe quel élément : un nœud final, un morceau de données dans une vidéo ou un livre, une commande pour allumer certaines lumières à distance…etc.

Les blocs de construction principaux de l'architecture NDN sont des blocs de contenu nommés, contrairement à l'unité de communication fondamentale de l'architecture IP, qui est un canal de bout en bout entre deux extrémités identifiées par des adresses IP.

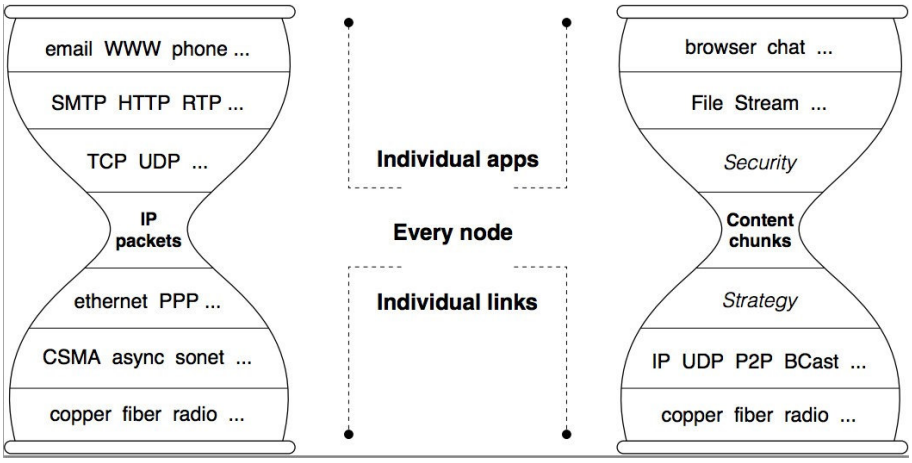

Figure 1.1: Comparaison entre architecture NDN et IP

[6] **CCN : C**ontent **C**entric **N**etworking.
[7] **UCLA : U**niversity of **C**alifornia at **L**os **A**ngeles**.**

1.3.3.1.Type de paquets manipulés [2]

La communication dans NDN est pilotée par des destinataires (consommateurs de données), par l'échange de deux types de paquets : l'intérêt (**interest**) et les données (**data**) comme l'indique la (**Figure I.2)**. Les deux types de paquets portent un nom (préfixe) qui identifie une donnée qui peut être transmise dans un paquet de données.

- **Le paquet Interest :** un consommateur met le nom d'une donnée désirée dans un paquet d'intérêt et l'envoi au réseau. Les routeurs utilisent ce nom pour transférer l'intérêt vers le producteur de cette donnée.
- **Le paquet Data :** une fois que l'Intérêt atteint un nœud qui a la donnée demandée, ce dernier enverra un paquet de données qui contient à la fois le nom de cette donnée et son contenu, ainsi qu'une signature numérique par la clé du producteur pour des raisons de sécurité. Ce paquet suit le sens inverse du chemin emprunté par le paquet d'intérêt pour revenir au demandeur.

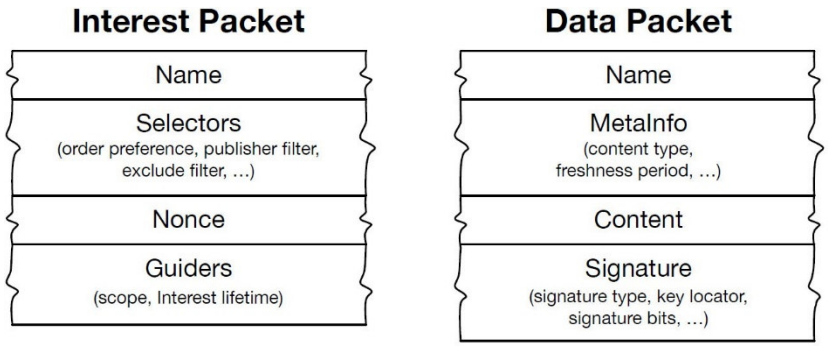

Figure 1.2: Les paquets NDN

- **Format de paquets NDN**

  - **Le format TLV**

TLV (type-length-value) est un moyen de stocker des données pour faciliter l'analyse rapide de ces données. Il est principalement utilisé pour transférer des données au format binaire dans les communications réseau.

Le type et la longueur sont fixés en taille (généralement entre 1 et 4 octets) et le champ de valeur est de taille variable. Ces champs sont utilisés comme suit :

- ***Type :*** Un code binaire, souvent simplement alphanumérique, qui indique le type de champ que représente le message.
- ***Longueur :*** La taille du champ de valeur (typiquement en octets).

- ***Valeur :*** Série d'octets de taille variable contenant des données du message.

➢ **Encodage de type longueur-valeur (TLV)**

```
NDN-TLV := TLV-TYPE TLV-LENGTH TLV-VALUE
TLV-TYPE := VAR-NUMBER
TLV-LENGTH := VAR-NUMBER
TLV-VALUE := BYTE
```

Le format de paquet NDN n'a pas d'en-tête de paquet fixe et qu'il n'encode pas un numéro de version de protocole. Au lieu de cela, la conception utilise le format TLV pour offrir la flexibilité d'ajouter de nouveaux types et de supprimer progressivement les anciens types à mesure que le protocole évolue avec le temps. L'absence d'un en-tête fixe permet de supporter efficacement des paquets de très petites tailles, sans le surcoût d'en-tête. Il n'y a pas non plus de support de fragmentation de paquets au niveau du réseau. Chaque fois que nécessaire, les paquets NDN peuvent être fragmentés et réassemblés saut par saut.

Le paquet Interest

Le format TLV du paquet Interest est sous la forme suivante :

```
Interest :: = INTEREST-TYPE  TLV-LENGTH
              Name
              Selectors ?
              Nonce
              Guiders ?
```

Le signe (?) défini que le champ est facultatif.

Name et Nonce sont les deux éléments importants et requis dans un paquet Interest. Les sélecteurs sont des éléments optionnels qui qualifient davantage les données pouvant correspondre à l'intérêt. Ils sont utilisés pour découvrir et sélectionner les données qui correspondent le mieux. Ces sélecteurs sont placés juste après le nom pour faciliter les implémentations.

➢ **Name :** champ obligatoire qui représente le préfixe de la donnée, un paquet doit avoir au moins un nom**.**

➢ **Selectors :**

```
Selectors ::= SELECTORS-TYPE TLV-LENGTH
              MinSuffixComponents ?
              MaxSuffixComponents ?
              PublisherPublicKeyLocator ?
              Exclude ?
              ChildSelector?
              MustBeFresh?
```

✓ *MinSuffixComponents, MaxSuffixComponents*

Si nécessaire, MinSuffixComponents et MaxSuffixComponents permettent à un consommateur de données d'indiquer si le nom dans l'intérêt est le nom complet, y compris le résumé, ou le nom complet, sans le résumé. Ces deux paramètres se réfèrent au nombre des composants de nom au-delà de ceux du préfixe qu'un consommateur pourra définir, et en comptant le résumé implicite, qui peut apparaître dans les données correspondantes.

La valeur par défaut pour MinSuffixComponents est 0 et pour MaxSuffixComponents est effectivement infinie, ce qui signifie que toute donnée dont le nom commence par le préfixe est une correspondance. Souvent, un seul d'entre eux sera nécessaire pour obtenir cette correspondance.

✓ *PublisherPublicKeyLocator*

Cet élément spécifie le nom de la clé utilisée pour signer le paquet de données que le consommateur demande. C'est un moyen pour l'intérêt de sélectionner les réponses (les paquets de données) d'un producteur particulier.

✓ *Exclude :*

Les sélecteurs Exclude permettent au demandeur de spécifier une liste ou des plages de composants de noms qui ne doivent pas apparaître comme une continuation du préfixe de nom dans le paquet de données répondant à l'intérêt. Par exemple, si l'intérêt est exprimé pour **/ndn/edu** et Exclude spécifie un composant de nom **ucla**, alors ni le producteur de données ni les routeurs NDN conformes ne sont autorisés à retourner un paquet de données ayant le préfixe **/ndn/edu/ucla**. Le filtre d'exclusion s'applique uniquement à un composant de nom du paquet Data.

✓ *ChildSelector :*

Souvent, un intérêt donné peut correspondre à plusieurs données dans un Content Store. Le ChildSelector fournit un moyen d'exprimer une préférence pour lequel de ces éléments devrait être retourné. Si la valeur est 0, l'enfant le plus à gauche est préféré. Si c'est 1, l'enfant le plus à droite est préféré. Ici, le plus à gauche et le plus à droite se réfèrent aux composants les moins importants et les plus grands selon l'ordre des composants du nom NDN.

Par exemple, en supposant dans la hiérarchie de noms, le composant qui vient immédiatement après le préfixe de nom est le numéro de la version, et le paramètre suivant étant le numéro de segment, alors le paramètre ChildSelector sera le numéro de version le plus à droite. Cependant, cette sélection est uniquement effectuée par rapport à un seul CS, pas globalement. Des tours supplémentaires qui excluent les versions antérieures peuvent être utilisés pour explorer d'autres CS pour les versions plus récentes. Dans ce cas, l'utilisation de ChildSelector ne modifie pas le résultat multi-round, mais diminue le nombre de rounds nécessaires pour converger vers une réponse.

✓ *MustBeFresh :*

Ce sélecteur est codé avec Type et Longueur mais pas de partie Valeur du format TLV. Lorsqu'il est absent d'un paquet d'intérêt, le routeur peut répondre avec un paquet de données de son CS dont la FreshnessPeriod (période de validité) est toujours valide ou expirée. Mais lorsqu'il est présent dans un paquet d'intérêt, le routeur ne doit pas renvoyer le paquet de données de son CS dont la FreshnessPeriod a expiré.

La FreshnessPeriod transportée dans chaque paquet de données est définie par le producteur d'origine. Il commence à décompter lorsque le paquet de données arrive à un nœud. Par conséquent, si un nœud est éloigné du producteur d'origine, il peut ne pas considérer l'état de données.

- **Nonce :** champ obligatoire qui contient une chaine de caractères de 4 octet généré aléatoirement. La combinaison de Name et de Nonce devrait identifier de manière unique un paquet d'intérêt, Ceci est utilisé pour détecter les intérêts en boucle.
- **Guiders :** il est facultatif et contient 2 champs qui affectent sur le comportement de transfert de l'intérêt, **Scope** et **InterestLifeTime**.

✓ *Scope :*

Cette valeur limite la propagation de l'intérêt comme suit :

- 0 : empêche la propagation au-delà du nœud local (même pour d'autres applications sur le même hôte).
- 1 : limite la propagation aux applications sur l'hôte d'origine.
- 2 : limite la propagation à pas plus loin que le prochain nœud.

Les autres valeurs ne sont pas définies à ce moment et entraîneront la suppression du paquet d'intérêt. Scope n'est pas un nombre de sauts : la valeur n'est pas décrémentée lorsque l'intérêt est transféré.

✓ *InterestLifetime :*

InterestLifetime indique le temps (approximatif) avant que l'intérêt expire. La valeur est en millisecondes. Le délai d'attente est relatif à l'heure d'arrivée de l'intérêt sur le nœud actuel.

Les nœuds qui transmettent des intérêts peuvent réduire la durée de vie pour tenir compte du temps passé dans le nœud avant le transfert, mais ne sont pas tenus de le faire. Il est recommandé que ces ajustements ne soient effectués que pour des retards relativement importants (mesurés en secondes). C'est l'application qui définit la valeur de l'**InterestLifetime**. Si l'élément **InterestLifetime** est omis, une valeur par défaut de 4 secondes est utilisée.

Le paquet Data

Le format TLV du paquet Data est sous la forme suivante :

```
Data :: = DATA-TLV  TLV-LENGTH
          Name
          MetaInfo
          Content
          Signature
```

Le paquet de données représente des données binaires dans son champ content accompagné du nom de la donnée, et quelques bits d'information supplémentaires (MetaInfo) ainsi qu'une signature numérique des trois autres éléments. La signature est placée à la fin du paquet pour faciliter l'implémentation car elle couvre tous les éléments.

- **Name :** même principe avec le paquet Interest.
- **MetaInfo :**

```
MetaInfo ::= META-INFO-TYPE TLV-LENGTH
ContentType ?
FreshnessPeriod ?
FinalBlockId ?
```

- ✓ ***ContentType :***

Trois ContentTypes sont actuellement définis : **Default** (= 0), **Link** (= 1) et **Key** (= 2). Le type de contenu par défaut (**Default 0**), qui correspond aux bits de données réels identifiés par le nom de données. Le type de contenu **Link** (**1**) est un autre nom qui identifie le contenu réel des données. Le type de contenu **Key** (**2**) est une clé publique.

- ✓ *FreshnessPeriod :*

Le paramètre **FreshnessPeriod** facultatif indique combien de temps un nœud doit attendre après l'arrivée de ces données avant de les marquer comme périmées. La valeur codée est le nombre de millisecondes. Les données périmées sont toujours des données valides, l'expiration de **FreshnessPeriod** signifie seulement que le producteur peut avoir produit des données plus récentes.

Chaque CS associe chaque élément de données à un bit de stabilité. Le réglage initial de ce bit pour le contenu nouvellement arrivé est "non périmé". Si les données portent **FreshnessPeriod**, alors après que les données ont été stockées dans le CS, elles seront marquées comme périmées.

Si un intérêt contient le **MustBeFresh**, une donnée dont le bit de stabilité est défini ne peut pas être envoyée en réponse à cet intérêt. Si une copie exacte d'une donnée obsolète arrive, l'effet est le même que si les données périmées n'avaient pas été présentes. En pratique, les données périmées doivent être classées en haut de la liste des éléments à supprimer du CS lorsqu'un quota de stockage a été atteint.

- ✓ *FinalBlockId :*

Le **FinalBlockId** facultatif indique l'identificateur du bloc final dans une séquence de fragments. Il devrait être présent dans le bloc final lui-même, et peut également être présent dans d'autres fragments pour fournir un avertissement avancé de la fin aux consommateurs.

- ➢ **Content :** c'est le contenu du paquet.

- **Signature :** La signature NDN est définie comme deux blocs TLV consécutifs : SignatureInfo et SignatureValue. Les considérations générales suivantes concernant les blocs **SignatureInfo** et **SignatureValue** qui s'appliquent à tous les types de signature :

  A. **SignatureInfo** est inclus dans le calcul de la signature et décrit entièrement la signature, l'algorithme de signature et toute autre information pertinente pour obtenir le certificat parent, tel que **KeyLocator**.
  B. **SignatureValue** est exclu du calcul de la signature et représente les bits réels et la valeur de la signature.

1.3.3.2. L'architecture du routeur

Chaque routeur NDN maintient trois (03) structures de données :

- **Table d'intérêt en attente (PIT) : P**ending **I**nterest **T**able (**PIT**) stocke tous les intérêts qu'un routeur a transmis mais pas encore satisfaits. Chaque entrée de la PIT enregistre le nom de la donnée porté dans l'intérêt, ainsi que ses interfaces entrantes (par lesquelles il est arrivé).
- **Base d'informations de transfert :** Forwarding Information Base (**FIB**) est une table de routage qui mappe les noms de données connus aux interfaces correspondantes. La FIB est peuplée par des protocoles de routage basés sur les préfixes de données, ou manuellement, et peut avoir plusieurs interfaces de sorties pour un préfixe.
- **Content store (CS) :** est un cache temporaire de paquets de données reçues par un routeur. Comme un paquet NDN est indépendant de l'endroit d'où il provient ou de son acheminement, il peut être mis en cache pour satisfaire de futurs intérêts, sans avoir recours à acheminer l'intérêt à chaque besoin vers le producteur.

1.3.3.3. Le processus de communication [3]

Lorsqu'un consommateur (demandeur de la donnée) désire une donnée spécifique, il envoie un paquet d'intérêt qui porte le nom de cette donnée. Ce processus est illustré par la (**Figure 1.3)**.

Ce paquet passe dans le réseau, et il est acheminé. Lorsqu'un nœud NDN le reçoit et une recherche dans son Content Store (CS) est effectuée en comparant le préfixe (le nom) du paquet d'intérêt reçu avec les préfixes existants dans le CS. Si une entrée existe, le paquet Data correspondant est renvoyé à l'interface demandeuse. Dans le cas où aucune correspondance n'a été retrouvée, Une recherche dans la PIT est effectuée sur le préfixe de l'intérêt pour savoir si

quelqu'un d'autre a demandé la même donnée et qui n'est pas encore servie. Si une entrée existe, alors l'interface d'arrivée de l'intérêt est ajoutée à cette entrée et le paquet est supprimé.

Si le routeur ne trouve pas une entrée PIT désirée, une recherche dans la FIB est effectuée pour cet intérêt. S'il y a une correspondance dans la FIB, le paquet d'intérêt est acheminé à la face correspondante dans l'entrée de la FIB et sinon le paquet est supprimé. Lorsqu'un routeur reçoit plusieurs intérêts pour une même donnée il ne transmet que le premier, et les autres sont ignorés tout en ajoutant leurs interfaces entrantes dans la PIT.

L'acheminement de ces intérêts se réalise par une stratégie précise dite stratégie d'acheminement qui aide le routeur à décider de l'action à faire pour un intérêt, par exemple : la suppression d'un intérêt en cas de congestion des liens, ou si l'intérêt est suspecté d'être la base d'une attaque DOS[8]. Cette stratégie cherche pour chaque intérêt l'entrée qui correspond au plus long préfixe dans la FIB afin que le transfert soit plus précis.

A l'arrivée d'un paquet de donnée (Data) à un routeur, ce dernier cherche dans la PIT l'entrée correspondante (l'intérêt qui porte le même nom que celui transmis dans le paquet data), et l'envoie ensuite vers toutes les interfaces sur lesquelles l'intérêt a été reçu, donc le paquet data prend le chemin inverse de l'intérêt. Après la transmission du paquet Data, le routeur supprime l'intérêt de sa PIT et met les données dans son CS, processus connu sous le nom du Caching.

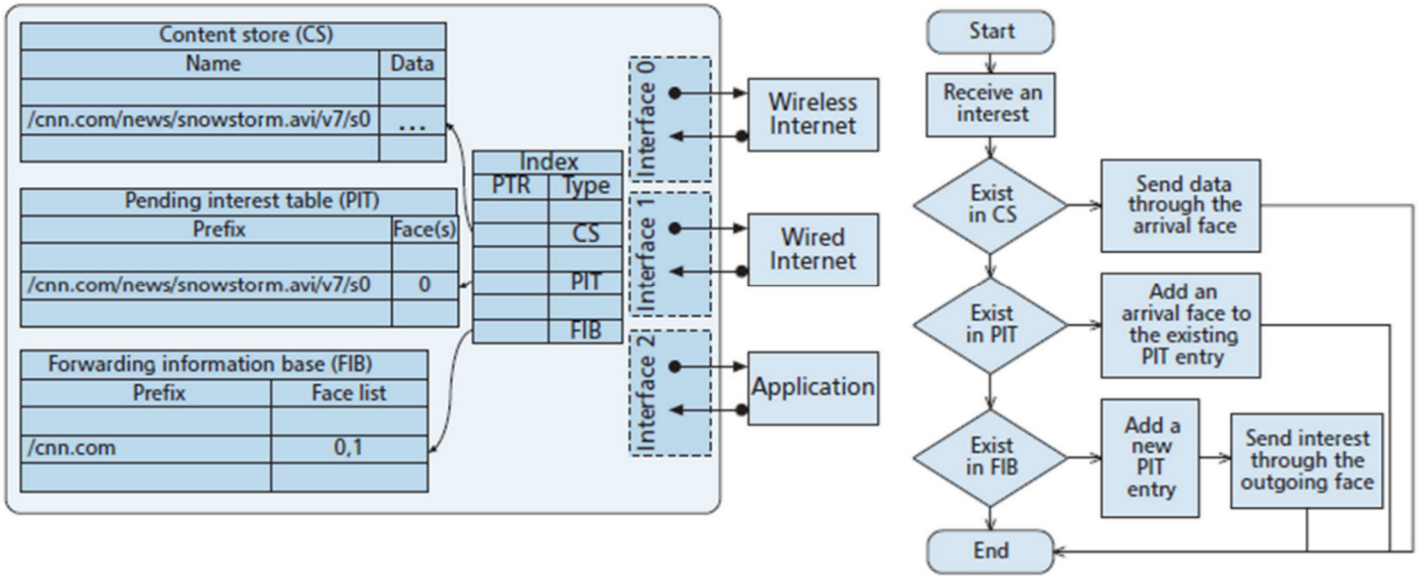

Figure 1.3: Processus de communication dans un nœud NDN [3]

[8] **DOS : D**enial **O**f **S**ervice : une attaque informatique ayant pour but de rendre indisponible un service.

Afin de récupérer les contenus volumineux qui comprennent plusieurs paquets, les intérêts contrôlent les flux d'une manière similaire que les ACK du protocole TCP, donc le paquet data pourrait être fragmenté en plusieurs pour respecter la MTU[9] qui reste la même dans les réseaux NDN tout comme les réseaux TCP/IP.

1.3.3.4. Le système de nommage NDN

Les noms attribués n'ont aucune signification dans le réseau, ils ont pour but l'identification des données et leur reconnaissances par les routeurs. Pour cela chaque application peut choisir un schéma de nommage qui lui correspond et qui sera différent des autres schémas et qui peut ainsi évoluer indépendamment du réseau.

La structure des noms NDN est hiérarchique, ces préfixes sont alors semblables aux **URLs,** exemple d'une vidéo sur le site youtube s'écrit sous la forme suivante : **/Youtube/video/exemple/vid.avi.** Cela permet aux applications de récupérer seulement quelques segments souhaités de cette vidéo, si une application désire le segment 3 de la vidéo **vid.avi**, le nom qui sera transmis dans le paquet d'intérêt sera ainsi : **/Youtube/video/exemple/vid.avi/3**

Finalement, cette structure hiérarchique offre le privilège de classer les préfixes dans un ordre comme une arborescence, et permet de les retrouver facilement en partant de la racine qui est **/youtube** pour notre exemple.

Pour récupérer les données, le consommateur doit être en mesure de construire d'une façon déterministe le nom de la donnée désirée sans l'avoir vu précédemment. Et cela se résume en deux possibilités :

- L'utilisation d'un algorithme déterministe par le producteur et ainsi par le consommateur, qui leurs permet d'arriver au même nom en se basant sur les informations disponibles à la fois.
- Le champ **selectors** du paquet d'intérêt pour lequel on associe le plus long préfixe, afin qu'il récupère la donnée souhaitée au bout de plusieurs itérations.

Pour la deuxième possibilité, un simple ensemble des sélecteurs peut s'occuper de la récupération de la donnée en connaissant partiellement son nom. Cela raffine la recherche et précise si la donnée est segmentée afin de récupérer facilement ces segments.

---

[9] **MTU :** La taille maximale d'un paquet pouvant être transmis en une seule fois (sans fragmentation) sur une interface.

1.3.4. Le caching NDN

Ces nouvelles architectures orientées contenu reposent sur un réseau de caches, ou toutes les entités du réseau (routeurs) possèdent une capacité de stockage pour conserver les données. Ce principe rappelle quelque peu les réseaux P2P[10] ou les contenus finiront par se propager au plus proches des utilisateurs ou être répliquées en plusieurs points du réseau et aucune donnée n'est associée a priori à une localisation précise. Ainsi, n'importe quelle entité du réseau pourra répondre à une requête si elle dispose du contenu désiré.

1.3.4.1. Le processus du caching

Étant donné que chaque paquet de données NDN est significatif indépendamment de l'endroit d'où il provient ou vers lequel il peut être transféré, un routeur peut le mettre en cache dans son Content Store pour satisfaire de futures demandes. Le content store est une mémoire cache. Il enregistre les copies de données, par conséquent un même contenu peut être stocké dans différent routeurs du réseau. Les routeurs NDN sont capables de réutiliser les données puisqu'ils sont identifiés par les noms de données.

La taille du content store est limitée et une fois qu'il est plein, une expulsion du cache est effectuée pour mettre de nouveaux éléments. Pour cela la gestion et le remplacement de cette mémoire cache sont soumis aux règles et politiques des fournisseurs de services. Ce processus est illustré par la (**Figure 1.4**).

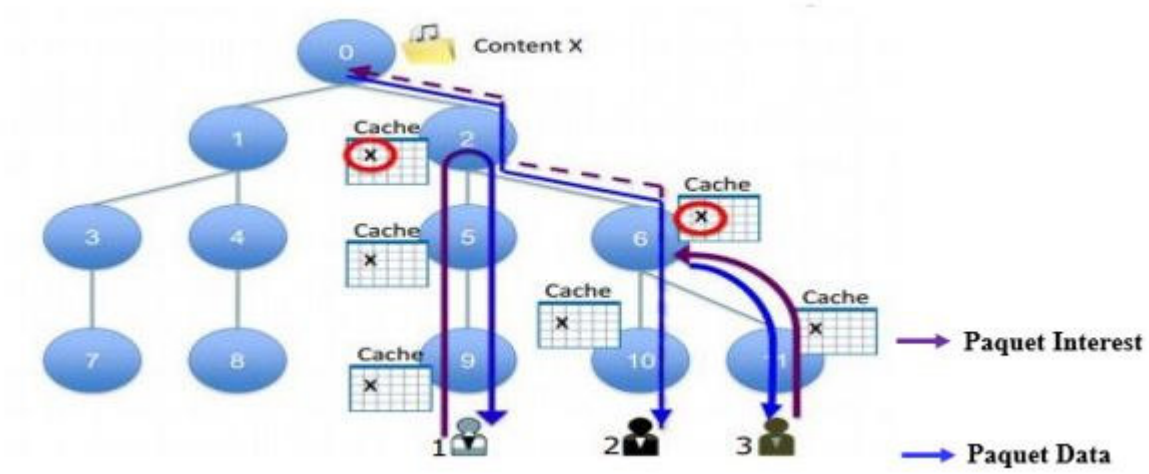

Figure 1.4: Le processus du caching dans les réseaux NDN

[10] **P2P :** est un modèle de réseau informatique où chaque entité du réseau est à la fois client et serveur contrairement au modèle client-serveur.

Selon la figure, le consommateur 2 désire une donnée X, donc il envoie une requête afin de la récupérer chez le producteur qui est le nœud 0. Ce dernier lui répond par un paquet Data et il sera mis en cache par les nœuds par lesquels il passe (2 et 6). Lorsque les consommateurs 1 et 3 demandent la même donnée X, la requête est satisfaite par les nœuds 2 et 6 respectivement, sans avoir recours au producteur puisqu'ils possèdent déjà la donnée dans leur content store.

1.3.4.2. Les objectifs du caching

Le caching permet aux nœuds de stocker les données qui transitent pour satisfaire les futures requêtes, voici quelques objectifs et avantages de ce processus :

- La réduction de la charge sur les serveurs et limiter l'impact d'une recherche dans le réseau.
- Le temps d'accès aux données est diminué car les contenus sont beaucoup plus proche des utilisateurs.
- Maintenir les requêtes d'intérêt en local permet d'éviter la surcharge du cœur du réseau, et de réduire la congestion.
- Le caching permet aussi d'optimiser la consommation de la bande passante, et la récupération rapide des contenus populaire et évitant le goulet d'étranglement d'un serveur unique.
- Si une perte due à la congestion survient, la mise en cache atténue l'impact puisque les intérêts retransmis peuvent être satisfaits par des paquets de données mis en cache juste avant le point de perte.

La mise en cache des données nommées peut soulever des problèmes de confidentialité. Les réseaux IP offrent une faible protection de la vie privée. On peut trouver ce qu'il y a dans un paquet IP en inspectant l'en-tête, et qui a demandé les données en vérifiant l'adresse de destination. Contrairement à l'IP, NDN nomme explicitement les données, ce qui permet sans doute à un moniteur réseau de voir quelles données sont demandées. Cependant NDN supprime entièrement l'information concernant qui demande les données. À moins qu'il ne soit directement connecté à l'hôte demandeur par une liaison point à point, un routeur saura seulement que quelqu'un a demandé certaines données, mais ne saura pas qui a émis la demande. Ainsi, l'architecture NDN offre naturellement une protection partielle de la confidentialité.

1.3.5. Le routage et l'acheminement dans NDN [4]

Nous avons étudié précédemment l'architecture NDN et discuté de son fonctionnement pour pouvoir remplacer l'architecture actuelle de l'internet. Le déploiement de cette architecture n'est toujours pas d'actualité car elle ne dispose pas de plan de routage adapté.

Le routage est la fonctionnalité qui va permettre aux nœuds du réseau de remplir leur table de transmission, qui leurs permettra ensuite d'acheminer les requêtes et les données correspondantes. En effet, dans l'architecture NDN, les messages adressent des noms et non plus des hôtes et cela modifie totalement la problématique du routage telle que connue pour l'Internet. De plus, les fonctionnalités de cache au cœur du réseau permettent de dupliquer les contenus et de dénombrer plus de copies dans le réseau. Ainsi, une requête vers un contenu pourrait être résolue via plusieurs nœuds pouvant répondre à cette requête. On ne peut donc plus se satisfaire des protocoles de routage utilises dans l'Internet actuel reposant sur la localisation des hôtes et des communications point-a-point et ce, afin de profiter pleinement de ces nouvelles fonctionnalités de mise en mémoire cache. Il convient donc de définir de nouveaux protocoles de routage adaptés aux spécificités de l'architecture orientée contenu Named-DataNetworking.

1.3.5.1. Les schémas de routage proposés pour NDN

- Open Shortest Path First for NDN
- Named-data Link State Routing
- Bloom filter-based Routing Approach
- Distance-based Content Routing
- Scalable Content Routing for Content-Aware Networking
- Routage basé sur les capacités de cache disponibles
- Controller based routing strategy for NDN
- SRSC : Protocole de routage basé sur SDN (Software-defined networking)

1.3.6. Relation entre le caching et le routage NDN

Les protocoles de routage proposés pour NDN, essayent d'exploiter au mieux le principe de caching. Les mémoires cache sont bénéfique dans le processus du routage, puissent qu'elles conservent les contenus déjà routés. En cas de nouvelle demande, la donnée sera directement acheminée de l'emplacement le plus proche du demandeur.

Grace au caching les protocoles de routages suppriment les redondances de données, si un paquet data arrive à un routeur, il garde une copie dans sa CS et à l'arrivée d'un nouveau paquet data pour la même donnée il sera supprimé.

En sachant que la quantité de donnée est énorme, et donc les mémoires cache des routeurs ont une grande taille bien qu'elles sont limitées, ce qui implique que le temps de parcours et d'accès aux donnée augmente.

1.4. Conclusion

Dans ce chapitre nous avons présenté l'architecture NDN, son fonctionnement et tous les mécanismes utilisés dans ce réseau. Ainsi que les types de paquets qui circulent et définir leur caractéristiques. En effet, les propriétés des noms et du système de nommage nous permettent d'exploiter au mieux les avantages de NDN et de faire une bonne conception dans les chapitres suivants.

L'architecture NDN prévoit plusieurs algorithmes de routages, soit des algorithmes déjà utilisé dans le réseau internet standard et personnalisé afin de pourvoir fonctionner sur NDN, soit la création de nouveaux algorithmes spécialement pour cette nouvelle architecture.

Après avoir expliqué l'architecture traitée dans ce mémoire, et la partie que nous devons implémenter (les caches NDN). Nous allons présenter dans le chapitre suivant le type de mémoire à utiliser pour ces caches, et définir leurs avantages et leurs inconvénients.

Chapitre 2 : Les mémoires TCAM

Chapitre 2 : Les mémoires TCAM

2.1. Introduction

Les systèmes informatiques sont de plus en plus présents dans notre quotidien, sous la forme téléphones portables, cartes à microprocesseur, ou autres PDA. Ces appareils sont constitués d'un ensemble formé de matériel et de logiciel qui prend aujourd'hui le nom de « système embarqué ». Ces systèmes sont particuliers. Ils ne correspondent pas à la vision que l'on peut avoir du conventionnel ordinateur personnel.

En effet, chacun de ces appareils est caractérisé par une ou plusieurs fonctions précises et a été conçu dans ce but. En particulier, ils répondent, matériellement à plusieurs besoins de l'utilisateur : communiquer avec d'autres appareils (par Bluetooth, wifi, Wimax[11]... etc), fournir une connexion à l'internet, savoir stocker diverses données.

Ces appareils respectent également des contraintes telles qu'une faible consommation énergétique, un faible coût de fabrication ou encore des dimensions réduites. Ces besoins et contraintes provoquent l'évolution rapide du matériel, afin de fournir toujours plus de fonctionnalités, et performances accrût (plus de mémoire pour stocker).

Un des composants essentiels de ces systèmes est la mémoire. Les mémoires sont généralement adressables, de façon que chaque octet soit numéroté par une adresse qui fais office d'identifiant permettant de sélectionner un octet particulier. Pour accéder à un octet bien précis, il suffit d'envoyer son adresse à la mémoire, et celle-ci se chargera de fournir son contenu sur le bus de données.

Mais certaines mémoires ne fonctionnaient absolument pas sur ce principe comme la mémoire cache d'un ordinateur. Elle fonctionne sur un principe de *Tags* [12]assez complexe et non sur des adresses mémoires. Les mémoires caches tiennent une place importante. Mais celles-ci ne sont qu'un sous-ensemble d'un type de mémoire bien plus grand. Il s'agit des **mémoires adressables par contenu**.

---

[11] **Wimax :** (**Worldwide Interoperability for Microwave Access**) désigne un standard de communication sans fil. Il est utilisé comme système de transmission et d'accès à Internet à haut débit, portant sur une zone géographique étendue.

[12] **Tags** : (ou étiquette) est un mot-clé associé à une information stockée dans une mémoire cache.

2.2. Les mémoires standards

Dans les systèmes embarqués, nous trouvons diverses informations à mémoriser. Nous distinguons les programmes qui doivent être exécutés par les processeurs et les données de l'application. Nous utilisons généralement plusieurs types de mémoires pour le stockage de ces informations **RAM**, **ROM**, **FLASH, CACHE**…etc.

Selon le type d'opération que l'on peut effectuer sur les données (Lecture, Écriture) nous pouvons distinguer trois types de mémoires. Les mémoires mortes (ROM) n'autorisant que les accès en lecture, utilisées pour stocker les informations nécessaires au démarrage et au fonctionnement d'un système. Les mémoires vives (RAM) permettant les accès en lectures et en écriture, et nécessitent une tension d'alimentation permanente pour assurer leur fonction de stockage car l'information mémorisée s'efface en absence d'alimentation. Les mémoires flash sont petites et réinscriptible, elles peuvent conserver les données enregistrées même lorsqu'elles ne sont plus alimentées.

2.3. Les mémoires adressables par contenus

2.3.1. Présentation des CAM [5]

CAM est l'acronyme de Content Addressable Memory dite **mémoire associative,** est un type particulier de mémoire à accès aléatoire spécialement conçue pour des recherches de mémoire extrêmement rapides mais très spécifiques, apparu en raison de la vitesse réduite de la RAM. CAM combine des cellules de stockage et des portes logiques de comparaison. Dans cette mémoire, l'opération de recherche est effectuée par le contenu plutôt que par l'adresse de l'emplacement de la mémoire. La comparaison de la clé de recherche est effectuée en parallèle avec tous les mots stockés, ce qui renvoie l'adresse de la meilleure correspondance car la caractéristique la plus importante dans une CAM est qu'une recherche d'une entrée peut être effectuée dans un cycle d'horloge unique.

Les mémoires associatives sont une solution matérielle au problème de la recherche d'un élément dans un ensemble de données. Le processeur envoie la donnée recherchée à la mémoire, et celle-ci va vérifier tous ses octets pour voir s'il y a correspondance. Une fois la donnée trouvée, la mémoire peut alors permettre de la modifier ou de la lire, ainsi que de récupérer son adresse : on tombe directement sur la donnée recherchée.

Grâce à ces mémoires, la complexité algorithmique de la recherche d'un élément dans un ensemble est très fortement diminuée et est de complexité constante. Elles permettent

d'effectuer des recherches en parallèle puisqu'elles traitent des données de grande taille et de différents formats. Les CAM peuvent être cascadées pour augmenter la taille des tables de recherche.

Malgré la rapidité et la flexibilité des mémoires CAM, elles présentent des inconvénients tels qu'un coût très élevé et une forte consommation d'énergie à cause du nombre important de circuits activés en parallèle.

2.3.2. Différence entre les mémoires CAM et les mémoires standards [6]

La vitesse d'une architecture traditionnelle de **von Neumann** est principalement limitée par le goulet d'étranglement entre la mémoire et l'unité centrale (CPU).

Une solution à ce problème consiste à éliminer la séparation entre le processeur et la mémoire en déplaçant les fonctions du processeur directement dans la mémoire. A ces fins, la mémoire adressable par contenu (CAM) est un moyen idéal pour réduire la distinction entre la mémoire et le traitement. Ainsi la recherche dans ces deux mémoire se différent comme illustré en (**Figure 2.5**) :

Dans les mémoires traditionnelles :

- L'entrée est l'emplacement de l'adresse du contenu qui nous intéresse.
- La sortie est le contenu de cette adresse.

Dans les mémoires CAM, c'est l'inverse :

- L'entrée est associée à une donnée stockée dans la mémoire.
- La sortie est l'emplacement où le contenu associé est stocké.

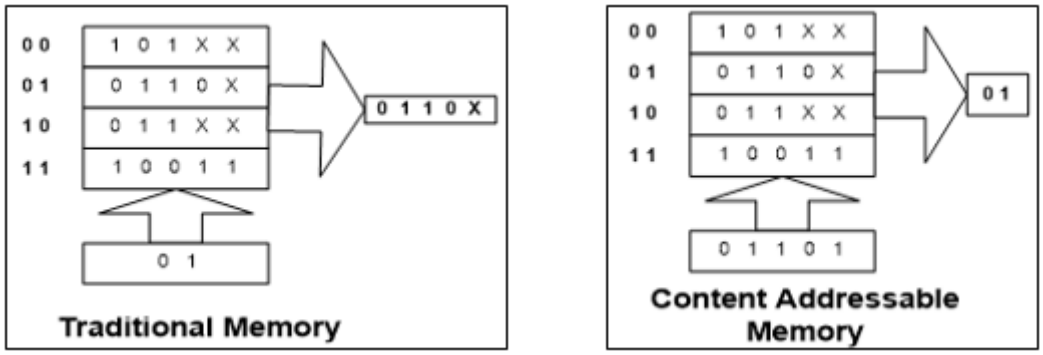

Figure 2.5: Différence entre CAM et mémoire standard

2.3.3. Les types de la mémoire CAM [7]

Il existe deux types des **CAM** : **BCAM** et **TCAM**.

2.3.3.1. Les BCAM

Les **BCAM** stockent et comparent des valeurs binaires, c'est à dire des '0' et des 'l'. Ce sont Les CAM les plus couramment implémentés .Ils ne recherchent que des uns et des zéros, une opération simple. La CAM binaire peut être utilisée uniquement pour les opérations de correspondance exacte. Les tables d'adresses MAC des commutateurs sont généralement stockées dans des CAM binaires. Tout commutateur est capable de transmettre des trames Ethernet aux lignes qui utilisent des CAM pour la recherche. S'ils utilisaient la RAM, le système d'exploitation devrait rechercher la correspondance dans toutes les locations possibles. Avec les CAM, le système d'exploitation peut trouver ce dont il a besoin en une seule opération.

2.3.3.2. Les TCAM

TCAM est l'abréviation de **Ternary Content Addressable Memory**, inclut des bits génériques « **X** » qui correspondent à la fois à un et à zéro. Ces caractères génériques peuvent être utilisés sur les deux opérations d'accès à la mémoire (indiquant que certains bits de la recherche sont "**don't care**"). La recherche entièrement parallèle est fournie par TCAM, donc plusieurs correspondances peuvent se produire et renvoie l'adresse de la meilleure correspondance. Cela facilite l'implémentation de nombreuses opérations complexes telles que la recherche dans les tables de routage des routeurs réseau. En raison du fait que la TCAM recherche chaque emplacement en mémoire en même temps, l'ordre des éléments dans la TCAM est moins important et les grandes structures d'indexation peuvent souvent être entièrement évitées et c'est ce qui rend le processus de recherche très rapide et à haute vitesse.

TCAM peut être implémentée dans plusieurs applications : les mémoires tampon de traduction dans les microprocesseurs, la compression de données, et dans d'autres opérations de réseau à grande vitesse telles que la classification de paquets, le contrôle de liste d'accès, les systèmes de détection d'intrusion, le filtrage en temps réel dans la détection de virus. Ainsi la recherche de motifs de gènes en bio-informatique et le traitement d'images. Mais l'application principale de TCAM est dans les systèmes de réseau où comparer l'adresse de destination du paquet entrant avec les adresses stockées et transmettre le paquet au port de sortie approprié.

L'architecture TCAM

L'architecture TCAM générale est constituée d'un réseau de mémoire centrale (cellules TCAM). Un décodeur d'adresse, un circuit de comparaison et un pilote de données **search-lines** (SL), des amplificateurs de détection ML et un encodeur de priorité tel représenté sur la (**Figure 2.6)**. Les mots mémoire sont disposés horizontalement, et constituent les données stockées dans la TCAM où chaque bit d'un mot est une cellule.

Pendant le processus de recherche dans la TCAM, l'entrée de ce système est le mot de recherche et qui est diffusé sur toutes les lignes de recherche, donc chaque mot stocké est comparé simultanément avec toutes les entrées. La ligne de correspondance (**Match-lines**) indique si le mot stocké est identique au mot recherché, si ces deux derniers sont identiques cette ligne sera activée afin d'indiquer qu'il y a une correspondance et son adresse sera envoyée. De plus, il y a souvent un signal dit miss qui signale le cas où il n'y a pas d'emplacement correspondant dans la TCAM.

Enfin dans le cas où plusieurs correspondances sont trouvées, l'encodeur logique de priorité hiérarchise les adresses et sélectionne l'emplacement correspondant le plus prioritaire à mapper, avec des mots dans des emplacements d'adresse inférieurs (la plus haute priorité). Cette stratégie est connue sous le nom **First Matching** (la première correspondance).

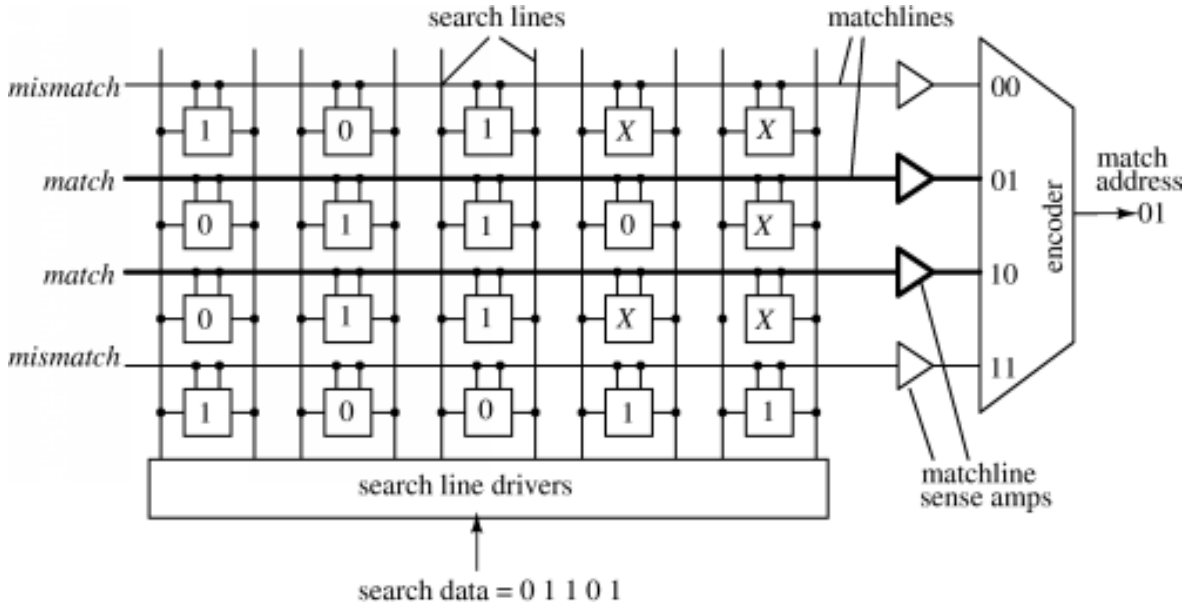

Figure 2.6: L'architecture TCAM

D'après cette figure, le mot recherché est (**01101**) et selon les données existantes dans la TCAM ce mot correspond aux deux lignes. L'encodeur (**prioritizer**) choisit donc l'emplacement le plus prioritaire avec la plus petite adresse entre **01** et **10** et renvoie alors **01**.

Une TCAM peut stocker et rechercher des états ternaires ('1', '0' et 'X'). L'état 'X', aussi appelé **'mask'** ou **'not care'**, peut être utilisé comme une entrée générique pour effectuer une correspondance partielle. Le masquage peut être effectué globalement (dans la clé de recherche) et localement (dans les entrées de la table). La (**Figure 2.7)** montre des exemples de masquage global et local dans les TCAM. La clé de recherche **10110XXX** correspondra à toutes les entrées qui se situent dans la plage suivante : '**10110000**'à '**10110111**' (mots situés aux adresses 1 et 4 dans ce cas). Il est appelé **masquage global** (**table (a)**) car les trois derniers bits de toutes les entrées de la table sont ignorés. Dans la **table (b),** le mot **110-XX-010** (situé à l'adresse **2**) correspond à l'une des clés de recherche suivantes : **110-00-010**, **110-01-010**, **110-10-010** et **110 -11-010**. Ce type alors est appelé **masque local**.

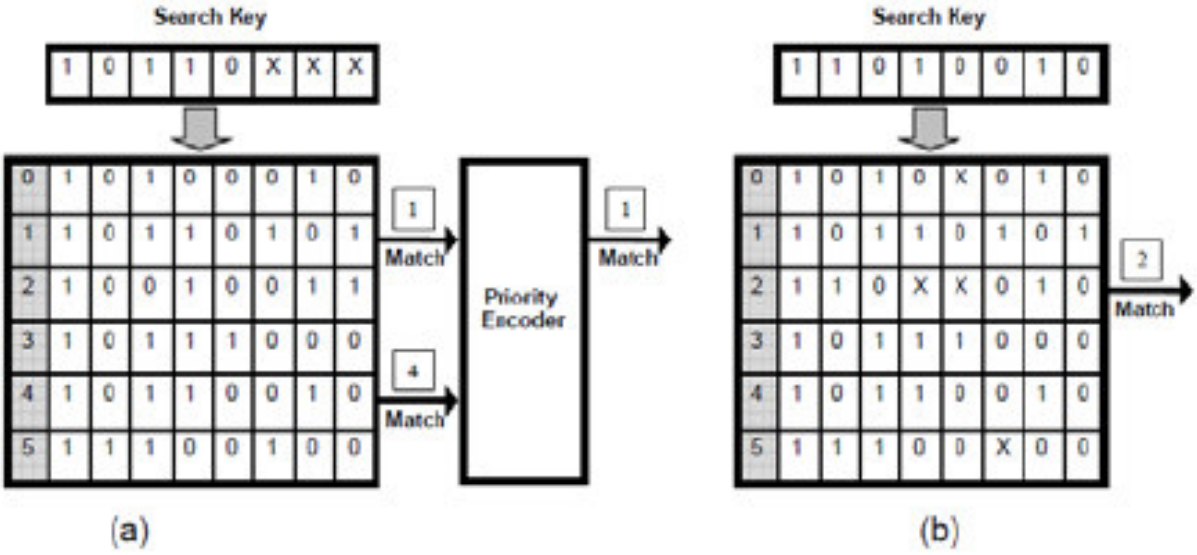

Figure 2.7: Opération de recherche dans un TCAM avec (a)- masquage global et (b)- masquage local

La cellule TCAM

Le plus grand avantage de l'utilisation d'une TCAM est que la comparaison se fait directement dans les cellules, ce qui signifie que pour comprendre la puissance consommée par l'opération de comparaison, il faut comprendre les internes d'une cellule TCAM. La consommation d'énergie élevée de TCAM est largement due à la fonction de comparaison utilisée dans l'opération de recherche. La cellule TCAM est constituée d'une partie de circuit de stockage et d'une partie de circuit de comparaison utilisée dans l'opération de recherche, comme représenté sur la (**Figure 2.8)**.

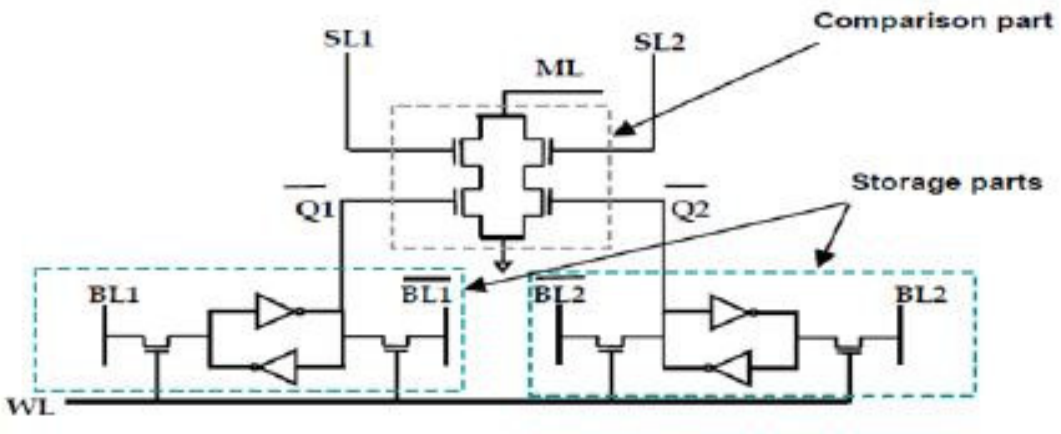

Figure 2.8: Cellule TCAM conventionnelle

Le noyau TCAM est implémenté comme un tableau de cellules TCAM (matrice) avec des lignes de mots mémoire (**WLs**) et des lignes de correspondance (**MLs**) **Match-lines** horizontales, et des lignes de bits (**BLs**) et des lignes de recherche (**SLs**) **Search-lines** verticales comme le montre la (**Figure 2.9)**. Toutes les cellules d'une même ligne partagent les même **WL** et **ML**, et toutes les cellules de la même colonne partagent les mêmes **SL**.

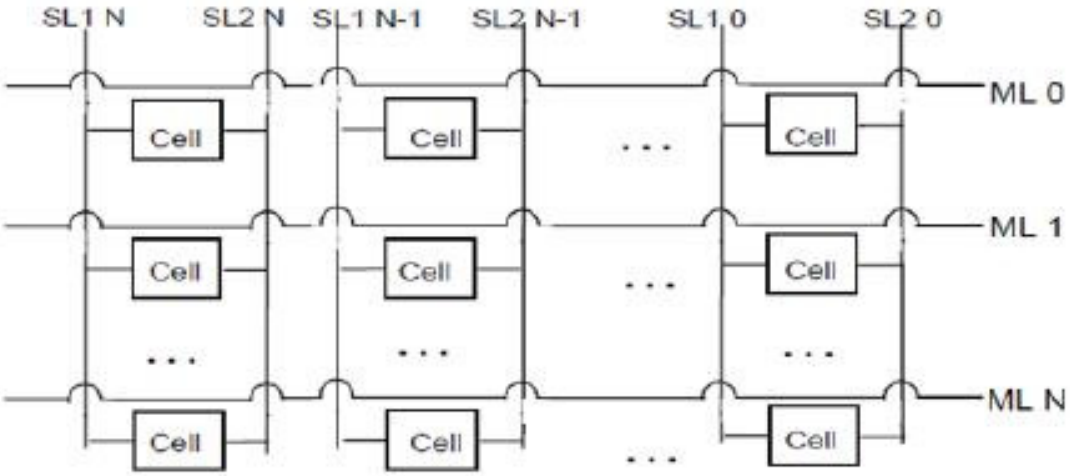

Figure 2.9: Noyau TCAM

2.3.4. Utilisation des mémoires CAM dans les réseaux IP [8]

2.3.4.1. Utilisation des mémoires BCAM

La CAM binaire effectue une recherche binaire et renvoie soit 1 soit 0. L'adresse MAC de destination est utilisée comme index pour rechercher le port de sortie. (Voir **Figure 2.10**)

1. Une trame Ethernet entre dans un commutateur.
2. L'adresse MAC de destination est extraite de l'en-tête Ethernet.

3. L'adresse MAC est déjà au format binaire et peut correspondre directement à chaque bit de la puce.
4. Ce processus fait appel à la table BCAM des adresses MAC, et trouve une correspondance

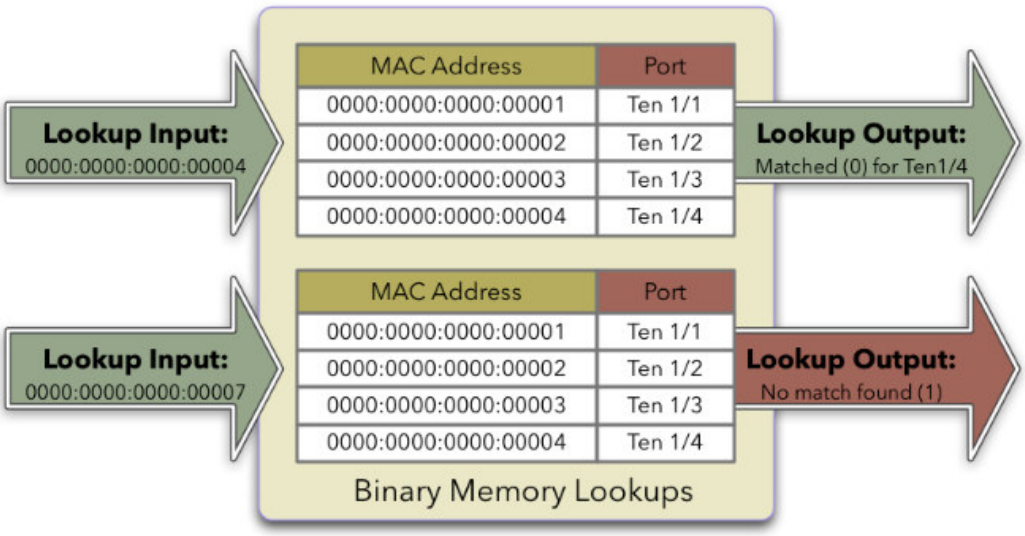

Figure 2.10: Exemple d'opération de CAM Binaire

Quand une correspondance se produit, la CAM binaire retourne la valeur du port de sortie et le paquet est mis en file d'attente pour la sortie sur ce port. Lorsqu'il n'y a pas de correspondance, le paquet sera envoyé sur chaque port du commutateur.

2.3.4.2. Utilisation des mémoires TCAM

Le matériel du réseau d'aujourd'hui utilise les couches 2, 3 et 4 du modèle OSI. Le périphérique peut basculer des trames Ethernet, router des paquets IP et implémenter des filtres basés sur les segments TCP. Une adresse MAC est toujours une correspondance exacte et utilise CAM binaire, mais la correspondance d'une route IP ou d'une liste d'accès nécessite une correspondance partielle. Un paquet IP aurait une adresse IP de destination spécifique de 103.23.3.7 correspond au préfixe 103.23.3/24 (l'adresse réseaux est 103.23.3.0/24) donc le prochaine saut est 171.3.2.22 (**voir la Figure 2.11**).

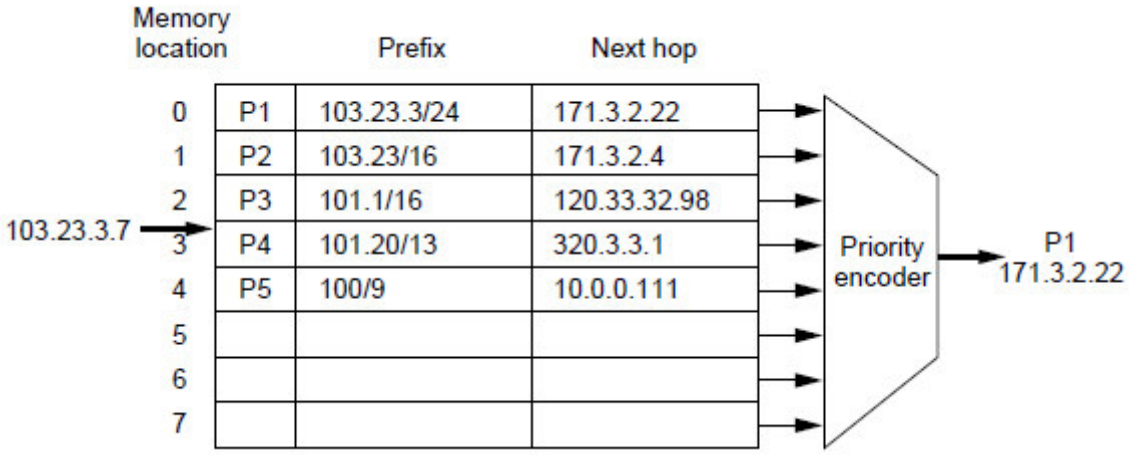

Figure 2.11: Exemple d'opération de CAM Ternaire

TCAM est particulièrement utile pour créer des tables de recherche sur les correspondances les plus longues dans les tables de routage IP. La table TCAM stocke ACL, QoS et d'autres informations généralement associées au traitement de couches supérieures.

La plupart des commutateurs possèdent plusieurs TCAM, de sorte que la sécurité entrante et sortante, ainsi que les ACL QoS, peuvent être évaluées simultanément ou entièrement en parallèle avec une décision d'acheminement de couche 2 ou de couche 3.

Cette opération de correspondance de préfixe la plus longue est effectuée dans un TCAM en stockant les entrées dans un ordre décroissant des longueurs de préfixe. Le TCAM recherche l'adresse de destination d'un paquet entrant avec tous les préfixes en parallèle. Plusieurs préfixes (jusqu'à L = 32 dans le cas de recherches IPv4) peuvent correspondre à l'adresse de destination.

2.3.5. Utilisation des mémoires TCAM dans le réseau NDN

Une caractéristique importante des réseaux NDN est la capacité de mise en cache de contenus directement dans les nœuds du réseau. Chaque contenu peut être mis en cache dans les nœuds de réseau se trouvant sur le chemin de la livraison de ce contenu, de sorte que les demandes ultérieures pourront être satisfaites plus rapidement, directement par ces caches. Les paquets perdus pourront aussi être récupérés plus rapidement par des retransmissions directes depuis les nœuds les plus proches. En suivant ce principe, les caches des routeurs NDN (Content Store) peuvent atteindre des milliers de lignes, et pour rechercher une donnée spécifique il faudra parcourir toute la table CS, donc il peut prendre beaucoup de temps qui augmente au fur et à mesure de nombre de lignes (taille) du CS.

Comme vu précédemment les mémoires **TCAM** ont un succès dans le monde de l'embarqué grâce à leur avantages tels que la vitesse de recherche qui se fait en un seul cycle d'horloge. Elles peuvent être considérées comme une meilleure solution matérielle pour ce type de réseau. Cela en implémentant la TCAM dans les Content Store des routeurs NDN.

Proposant maintenant une solution pour les routeurs NDN, par l'implémentation des deux table **CS** et **PIT** à l'aide d'une TCAM. Pour cela il faudra créer un autre composant qui servira d'un gestionnaire pour gérer les différentes activités de recherche qui se font dans le Content Store et la table **PIT**.

2.4. Conclusion

Dans ce chapitre, nous avons essentiellement classifié les mémoires selon l'opération que nous pouvons effectuer sur les données (lecteur, écriture), tout en décrivant brièvement les types des mémoires qui existent dans les systèmes embarqués : mémoires mortes et vives, les caches ainsi les mémoires adressable par contenu ou nous avons comparé ce type de mémoire avec les mémoires standard et tirer quelques différences. Nous avons fini par la présentation des mémoires **CAM** (**TCAM** et **BCAM**) où nous avons cité leurs caractéristiques et leurs fonctionnements. Et pour conclure, on a présenté la solution que nous pouvons retenir pour le réseau NDN qui est la mémoire TCAM.

Le prochain chapitre se charge de la présentation de nos contributions, l'analyse et la conception seront traitées par les sections à suivre.

Chapitre 3 : Conception

Chapitre 3 : Conception

3.1. Introduction

Le principe de réseau **NDN** (**Named Data Networking)** est que L'internet est principalement utilisé comme un réseau de distribution d'informations permettant aux utilisateurs de se concentrer sur les données dont ils ont besoin, plutôt que de devoir se référencer à un emplacement physique spécifique. Cela consiste en la diffusion de données d'une source à un certain nombre d'utilisateurs pour une mise en cache de contenu, permettant de réduire la congestion et d'améliorer la rapidité de livraison, simplifier la configuration des périphériques réseau et de renforcer la sécurité au niveau des données dans le réseau. Cependant, avec la quantité de donnée circulante sur internet qui augmente de plus en plus, le fait de sauvegarder beaucoup de données dans les routeurs augmentera le temps d'accès à ces données surtout que la recherche se fait grâce à des noms longs.

Ce chapitre consiste à concevoir un circuit implémentant quelques fonctionnalités relatives au routeur NDN, notamment le caching**,** avec des mémoires plus performantes qui nous permettront une meilleure gestion de données, et cela en parcourant des milliers de lignes de données en un seul cycle d'horloge grâce à des mémoires spéciales appelées **TCAM**.

Afin de mettre en œuvre tout cela, nous avons procédé en premier temps à la réalisation des tables **TCAM** : **CS** (Contente Store) qui permet la sauvegarde de donnée et **PIT** pour la sauvegarde des intérêts, puis de passer vers le contrôleur qui va intégrer ces deux tables **TCAM** ainsi que le module de communication qui nous permettra l'échange de données entre notre routeur et un utilisateur externe via une communication série de type **SPI**.

3.2. Conception TCAM

Grâce à leurs rapidités pour faire face au problème de temps d'accès aux mémoires dans les routeurs NDN, nous avons pensé à implémenter les mémoires TCAM à la place des mémoires standards.

3.2.1. L'architecture proposée pour la TCAM

Notre architecture proposée pour la TCAM est décrite dans la **Figure 3.12**

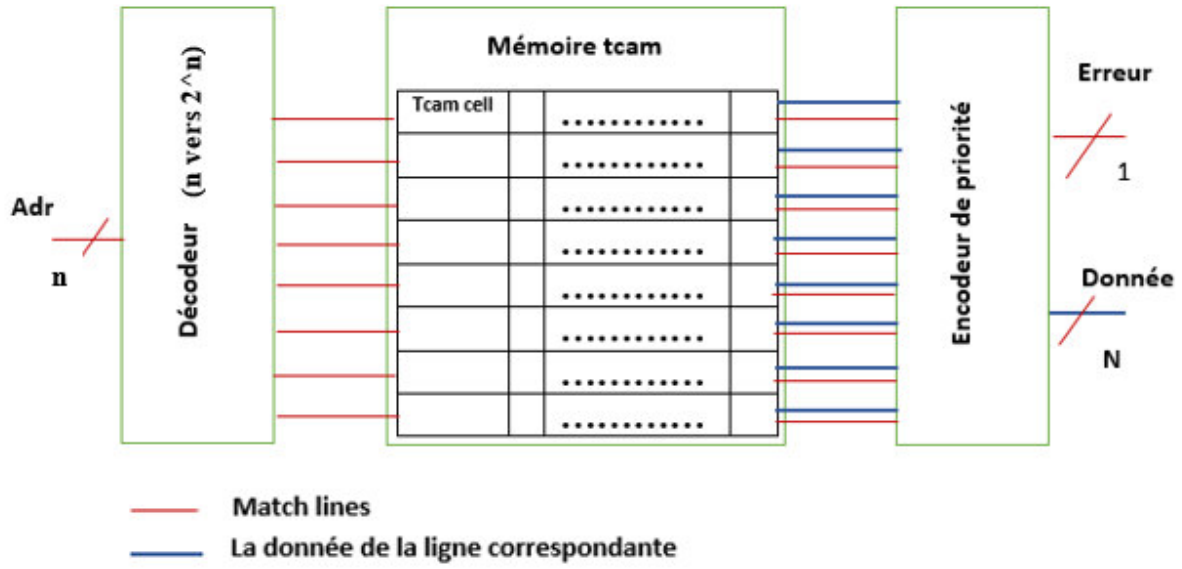

Figure 3.12: L'architecture proposée pour la mémoire TCAM

Notre circuit se compose de deux **TCAMs** qui sont la **CS** et la **PIT** chacune est constituée de trois parties qui sont **Décodeur** – la mémoire **TCAM** – **Encodeur de priorité.** Deux opérations peuvent être réalisées sur la TCAM : **l'écriture** et la **lecture.** L'écriture est faite en sélectionnant une ligne précise à l'aide du décodeur telle une mémoire ordinaire (RAM… etc), contrairement aux mémoires standards, la recherche dans la TCAM est réalisée en un seul cycle d'horloge et elle est en mesure de retourner la donnée recherchée grâce à notre implémentation matérielle.

Voici la conception détaillée de chacun de ces trois composants :

3.2.1.1. Le décodeur

Un décodeur est un circuit élémentaire dans chaque conception de circuit mémoire .Un décodeur y décode l'adresse et active la ligne correspondante. L'entrée **'Adr'** est codée sur un nombre spécifique de bits selon le nombre de lignes de la mémoire.

Notre mémoire **TCAM** contient 8 lignes (registres) pour cela on a besoin d'un décodeur 3 vers 8 pour les adresser.

La sortie est sur 8 bits où un seul bit qui est à **'1'** à la fois et le reste c'est des **zéros** .Ce circuit nous permet de spécifier la ligne où la donnée sera écrite.

3.2.1.2. La mémoire TCAM

Cette mémoire prend en entrée une donnée (paquet data ou paquet d'intérêt), et elle vérifie si celle-ci est présente. Une fois la donnée est trouvée, elle agit selon le fonctionnement de chacune des deux tables **TCAM** (**CS** et **PIT**). La recherche d'une donnée s'effectue donc en deux étapes : trouver la case mémoire qui contient la donnée, et ensuite faire le traitement qu'il faut (renvoyer la donnée elle-même en cas d'une CS, ou un signal d'erreur de recherche …etc.).

Notre bloc mémoire TCAM contient des paquets de donnée en cas d'un **Content store** et des paquets d'intérêts (le champ **Name** et le bit correspondant à l'interface d'entrée) en cas d'une **Pending Interest Table**.

Pour réaliser cette mémoire, on a fixé la taille de la **TCAM** à 8 x 512 bits pour **CS** et 8 x 260 bits pour la **PIT**. Pour cela on a utilisé 8 registres 512 bits pour **CS** et 260 bits pour la **PIT**. Donc on procède comme suit :

- **Bascule D** : Le composant qui nous permet de sauvegarder un bit (‘**0**’ ou ’**1**’ Logique).
- **Comparateur** : Une **TCAM** fait une recherche avec une donnée qu'on va lui fournir, et pour vérifier que la donnée à rechercher et celle enregistrée dans la **bascule D** sont égaux, on aura besoin d'un comparateur.
- **Cellule TCAM** : Ce composant est le résultat de la combinaison des deux composants précédents pour former une cellule qui nous permet de sauvegarder un bit et de le comparer à une donnée reçue et de nous indiquer le résultat de la comparaison **(‘1’ : égaux**, **‘0’ : Non égaux**).
- **Mémoire TCAM** « **Content store** **»** : Une fois la cellule **TCAM** est réalisée, à base de celle-ci on réalisera le registre à 512 bits.
- **mémoire TCAM** « **Pending Interest Table»** : Une fois la cellule **TCAM** est réalisée, à base de celle-ci on réalisera le registre à 260 bits.

3.2.1.3. L'encodeur de priorité

L'encodeur reçoit les matchs (les lignes de correspondance) de toutes les lignes ainsi toutes leur données et selon la ligne activée il retourne la donnée correspondante.

3.3. Circuit de gestion

Les deux tables (CS et PIT) qui sont des mémoires TCAM, sont accessibles via un systéme de communication série de type SPI. Ces mémoires ont un fonctionnement varié et multiple comme la recéption de la donnée à partir d'une communication avec le module externe, la recherche de la donnée, pour enfin renvoyer une correspondance en sortie ou bien un signal d'erreur. Toutes ces manipulation sur ces modules doivent être bien gérés, et pour cela nous proposons un circuit de gestion dédié à toutes ces fonctionnalités (un contrôleur).

3.3.1. Principe de fonctionnement

Comme nous l'avons vu dans le chapitre precedent, un réseau NDN, traite les données différement des réseaux classiques.

Deux paquets circulent dans le réseau ; **Data** (venant d'un serveur ou d'un routeur voisin) et **Interest** (venant du client ou d'un routeur voisin). Le fonctionnement est décrit et illustré par la (**Figure 3.13)**.

Lorsqu'un paquet arrive l'algorithme identifie d'abord son type :

- Si le paquet est de type **Data** le routeur cherche une correspondance dans la **PIT** pour satisfaire les précedantes demandes pour cette donnée. Et si une entrée existe il l'enregistre dans son cache (**content store**) afin de servir de futures demandes. Sinon le paquet **Data** est ignoré.
- Si le paquet est un **interest** le routeur cherche la donnée correspondante dans le cache **CS.** Si elle existe, il la retourne vers l'interface qui l'a demandée .
- En cas d'absence de la donnée dans **CS,** l'algorithme effectue une recherche dans la **PIT** en utilisant son nom. Si elle existe, donc il y a eu déjà une demande de cette donnée, il suffit juste d'ajouter le numéro de l'interface entrante .
- Si cet interest n'existe pas dans la PIT, il sera enregisté et ensuite il sera diffusé vers toutes les interfaces.

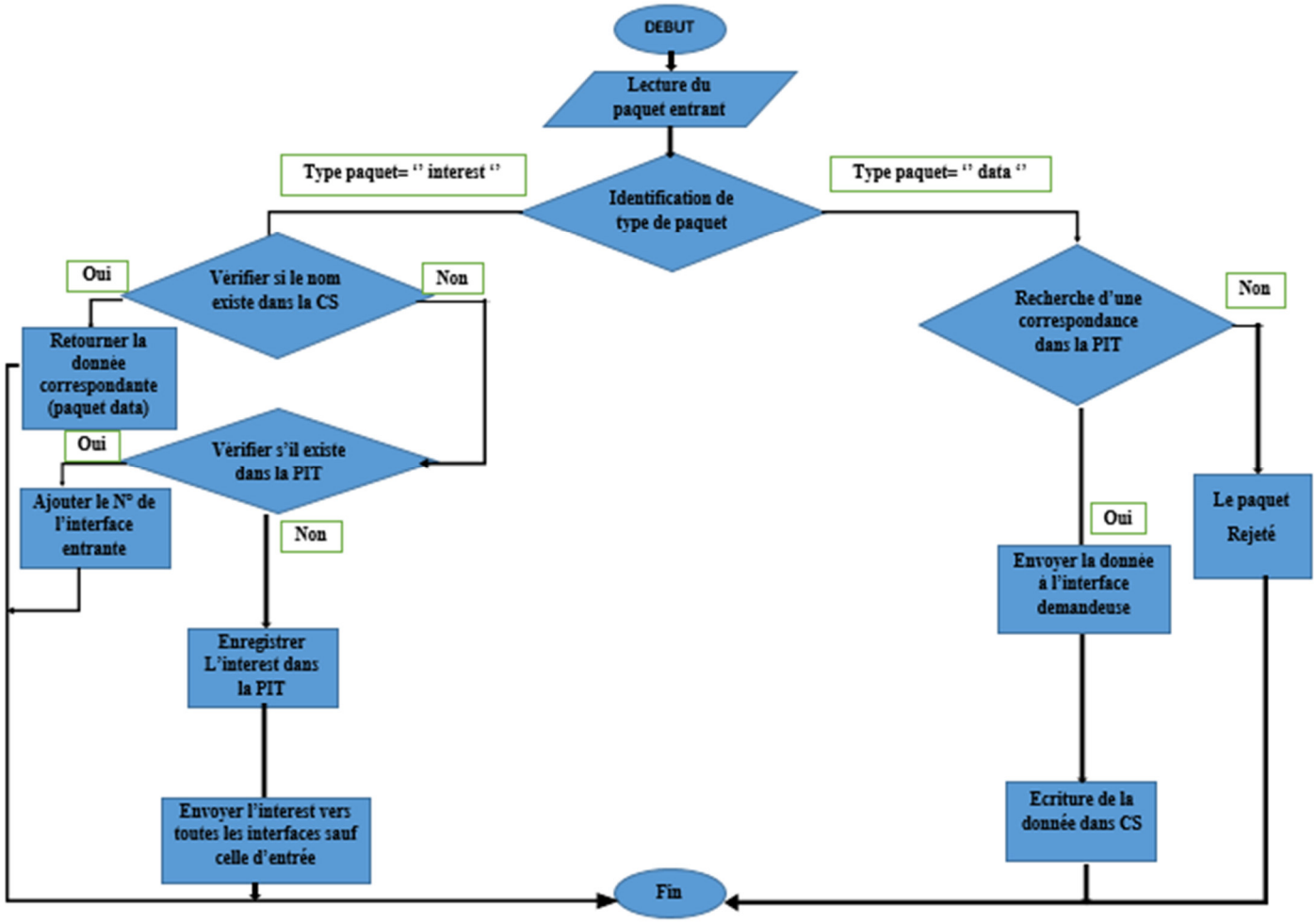

Figure 3.13: Organigramme explicatif du fonctionnement d'un routeur NDN

3.3.2. Interface externe du circuit

3.3.2.1. La communication série [9]

Les bus série deviennent de plus en plus courants. La communication série est utilisée pour toutes les communications longue distance et la plupart des réseaux informatiques, où le coût des problèmes de câblage et de synchronisation rend la communication parallèle impraticable. Une connexion en série occupe moins d'espace. L'espace supplémentaire permet une meilleure isolation du canal de son environnement. Les liens série peuvent être synchronisés beaucoup plus rapidement que les liens parallèles afin d'obtenir un débit de données plus élevé. La diaphonie est moins problématique, car il y a moins de conducteurs à proximité. Dans de nombreux cas, la communication en série est une meilleure option car elle est moins chère à mettre en œuvre. De nombreux circuits intégrés ont des interfaces série, par opposition aux

circuits parallèles, de sorte qu'ils ont moins de broches et sont donc moins chers. C'est pour cela qu'on a opté pour la liaison série SPI comme moyen d'interaction avec notre circuit

3.3.2.2. Protocole de communication SPI [9]

Le bus SPI est une norme de liaison de données série synchrone, nommée par Motorola et fonctionnant en mode duplex intégral. Les appareils communiquent en mode maître / esclave où le dispositif maître initie l'horloge et la trame de données. Les périphériques esclaves multiples sont autorisés avec des lignes de sélection d'esclave individuelles (sélection de puce « **Chip Select** ») comme montré dans la (**Figure 3.14**). SPI est souvent appelé SSI (Synchronous Serial Interface). L'interface de bus SPI communique en utilisant deux lignes de données (**MOSI** et **MISO**), une ligne de contrôle (**SS**) et une horloge de synchronisation, à savoir l'horloge série (**SCLK**), comme c'est définit :

- **Master out Slave in (MOSI) :** Sortie des données du maître vers les entrées des esclaves.
- **Master in Slave out (MISO) :** Sortie des données d'un esclave à l'entrée du maître.
- **Horloge série (SCLK) :** Horloge commandée par le maître aux esclaves, utilisée pour synchroniser les bits de données.
- **Slave Select (SS) :** Sélectionne le signal piloté par le maître vers des esclaves individuels, utilisé pour sélectionner l'esclave cible.

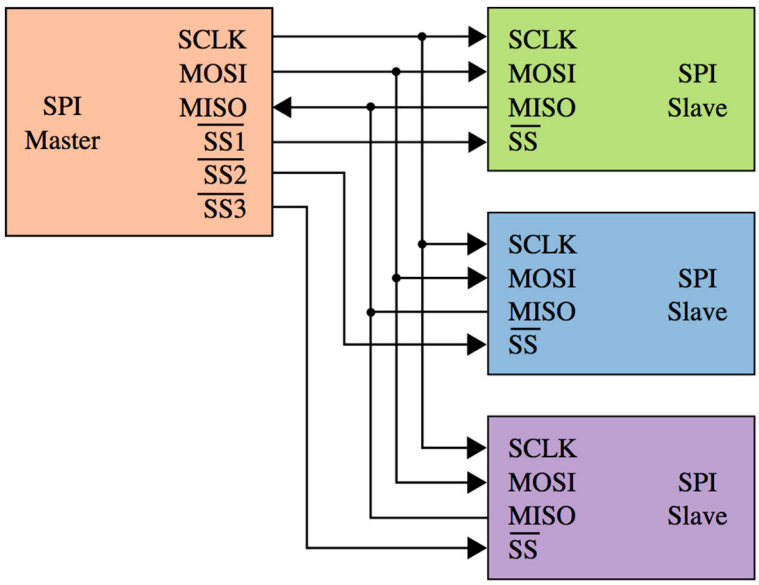

Figure 3.14: Communication SPI - un maître et trois esclaves.

Pour commencer une communication, le maître du bus configure d'abord l'horloge en utilisant une fréquence inférieure ou égale à la fréquence maximale prise en charge par l'équipement esclave. De telles fréquences sont généralement dans la plage de 1 à 100 MHz, le maître transmet alors le bit de **chip select** approprié pour l'esclave désirée à la logique **0**. La logique **0** est transmise parce que la ligne de sélection est active au bas niveau. Si une période d'attente est requise, le maître doit attendre au moins cette période avant de commencer à émettre des cycles d'horloge.

Les transmissions impliquent normalement deux registres à décalage d'une taille de mot donnée, tels que huit bits, un dans le maître et un dans l'esclave, ils sont connectés dans un anneau. Les données sont généralement décalées en premier avec le bit le plus significatif, tout en décalant un nouveau bit le moins significatif dans le même registre. Après que le registre a été déplacé, le maître et l'esclave ont échangé des valeurs de registre. Ensuite, chaque appareil prend cette valeur et l'écrit en mémoire. S'il y a plus de données à échanger, les registres à décalage sont chargés avec de nouvelles données et le processus se répète. Les transmissions peuvent impliquer n'importe quel nombre de cycles d'horloge. Lorsqu'il n'y a plus de données à transmettre, le maître arrête de basculer son horloge. Normalement, il désélectionne alors l'esclave.

3.3.2.3. L'architecture de communication proposée

La communication série proposée est une communication half-duplex (l'envoie et la réception ne se font pas au même temps) entre deux composants : Le circuit TCAM (implémenté sur une carte FPGA) est considéré comme esclave et un client (implémenté sur une carte Arduino) est considéré comme le maître qui initie la transaction.

Le circuit TCAM (FPGA) a un module qui implémente les fonctionnalités d'envoie et de réception, et ce dernier est relié à un gestionnaire qui permet de gérer le processus de communication d'une manière automatique avec le client (Arduino). Cette partie explique les différents signaux utilisés :

- **Module de communication du routeur NDN (FPGA) :** deux fonctionnalités (modules) sont implémentées, le processus de transmission et celui de réception servant de communication SPI avec l'utilisateur Arduino en utilisant trois signaux entrant (**G_SCLK, G_RX, G_AR_FP**) et trois autres sortants (**G_TX**, **G_CC, G_FP_AR**), et possédant ainsi deux registres à décalage (**TX** pour la transmission **et RX** pour la réception), expliquons maintenant l'intérêt et l'utilité de chaque signal :

Les signaux entrants :

- **G_SCLK :** est un signal d'horloge moins rapide que l'horloge de FPGA (division de fréquence de l'FPGA), envoyé par l'utisateur arduino afin de synchroniser la communication avec l'esclave.
- **RX :** représente un registre à décalage à gauche, qui permet de stocker la donnée reçu sur l'entrée **G_RX** en décalent les données bit par bit.
- **G_RX :** signal entrant vers FPGA (routeur NDN) qui est relié à la ligne de transmission (**TX**) d'Arduino, et a comme fonction la réception série de la donnée (bit par bit).
- **G_AR_FP :** signal venant de la carte Arduino vers FPGA, l'informant du début de la transmission et de sa fin. Lorsque **G_AR_FP = '1'** la carte Arduino est en plein processus de transmission vers Le routeur, et quand c'est égale à **'0'** c'est donc la fin de ce processus et le circuit TCAM (FPGA) peut désormais récupérer les données reçues en série par la ligne **G_RX**.

Les signaux sortants :

- **TX :** représente un registre à décalage à gauche, qui permet stocker la donnée à envoyer et de la sérialiser en la décalant, un seul bit à la fois est transmis sur la ligne **G_TX**.
- **G_TX :** signal sortant vers Arduino qui représente la ligne de transmission de la donnée en série (bit par bit), elle est reliée vers la ligne **RX** de l'utilisateur.
- **G_FP_AR :** signal mis en sortie pour informer l'utilisateur du début de la transmission de donnée par le routeur quand elle est à **'1'** et aussi de la fin quand elle est à **'0'**.
- **G_CC :** représente la ligne de sélection (**chip select**) mise à zéro par le circuit TCAM afin de sélectionner la carte Arduino désirée en communication.

L'utilisateur (Arduino) contient un module complet qui englobe les deux fonctionnalités (envoi et réception) gérée par des signaux semblable à ceux du routeur qui sont d'ailleurs reliés à ces derniers comme illustré par la (**Figure 3.15)**.

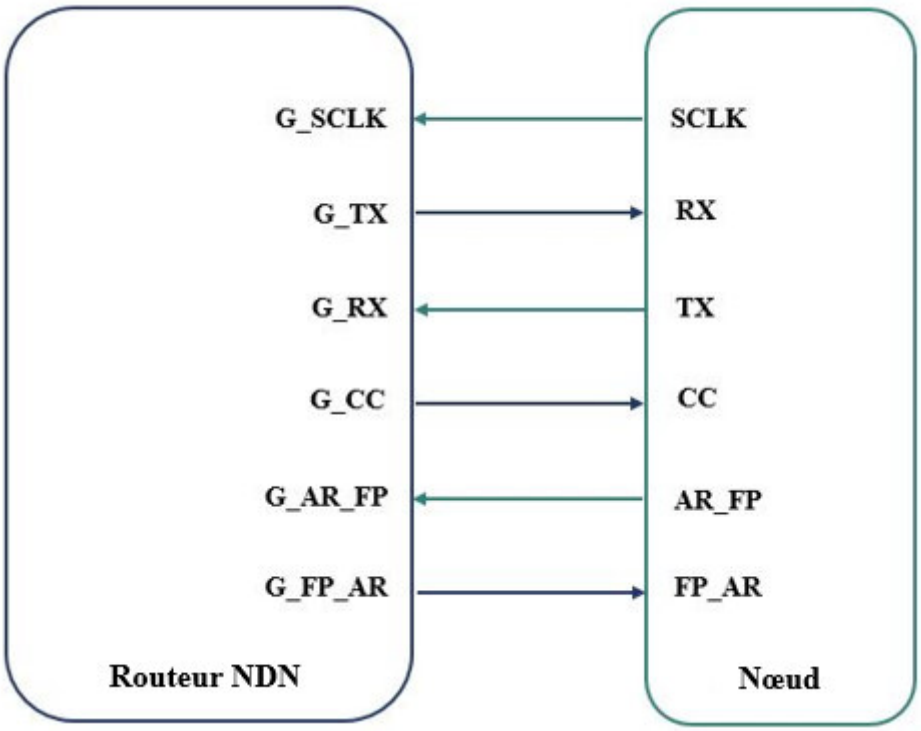

Figure 3.15: Interface de communication SPI proposée

3.3.3. Interface interne du circuit

La structure interne du circuit de gestion est décrite par la (**Figure 3.16)**. Il est composé de quatre modules dont deux entre eux sont complémentaires et permettent de gérer la communication avec l'extérieur, via une interface d'entrées / sorties. Les autres modules sont des mémoires TCAM (Content store et Pending interest table).

Le contrôleur (gestionnaire) est l'organe responsable de toutes les opérations du circuit. Ce gestionnaire englobe tous les modules et gère les interactions entre eux. Expliquant à présent ce processus de gestion.

En premier temps, il gère d'abord l'interface d'entrées / sorties d'où il reçoit les données de l'extérieur (clients ou routeur voisin). Ensuite l'unité (module) de réception reçoit la donnée en série (bit par bit) à l'aide d'un registre à décalage interne, et la sauvegarde dans le registre réception qui est ensuite acheminé vers les mémoires TCAM (soit la CS ou la PIT) pour une écriture ou une lecture. Si c'est une lecture, cela se résume en une recherche d'un nom de donnée dans les deux TCAM (selon l'algorithme décrit dans la section précédente) et en cas d'une demande on retourne la donnée désirée si elle existe (dans la CS) et la faire passer au registre d'envoie et l'unité d'envoie s'occupe de son acheminement vers le demandeur (client ou routeur voisin).

Si c'est une écriture, le gestionnaire s'occupe de la vérification de l'état des mémoires (si elles sont pleines donc pas d'espace disponible), et dans le cas où toutes les conditions d'une écriture sont vérifiées il écrit la donnée dans la mémoire correspondante.

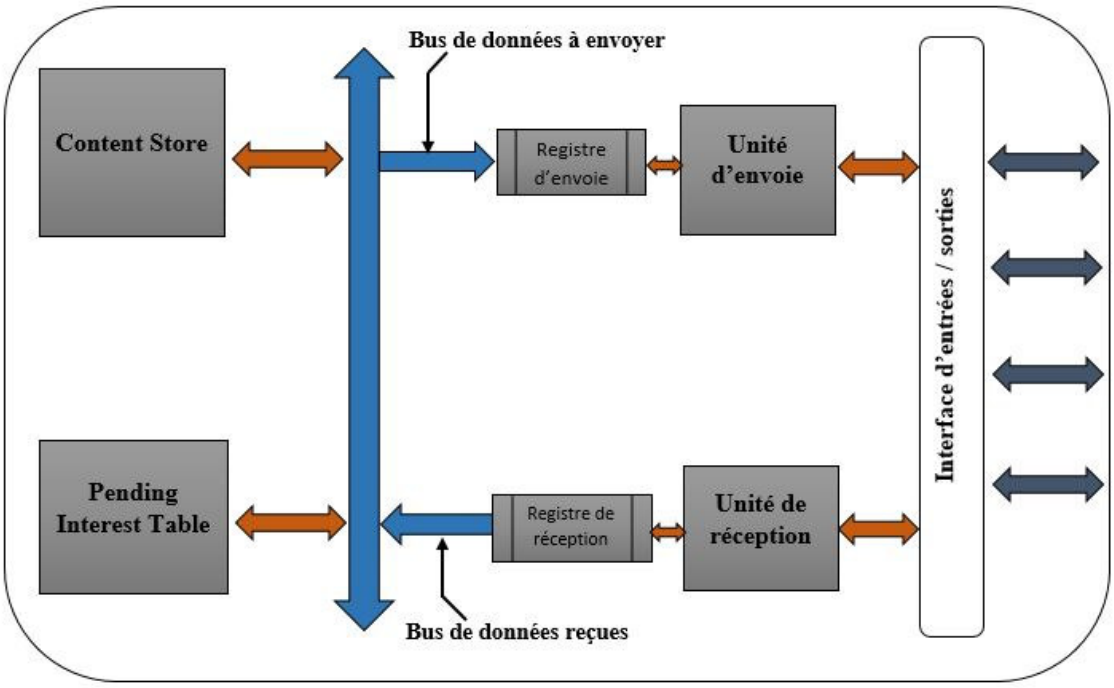

Figure 3.16: Architecture interne du circuit de gestion (Contrôleur)

3.3.4. Interface utilisateur du circuit

Notre circuit implémente les fonctionnalités de caching d'un routeur NDN qui communique avec les utilisateurs (client final) ou d'autres routeurs voisins, cette communication se fait via des échanges de paquets. La (**Figure 3.17)** décrit la structure proposée pour ces paquets dans notre projet.

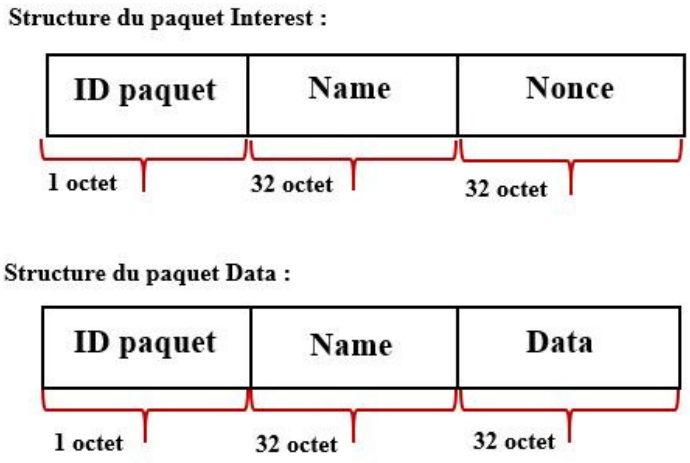

Figure 3.17: Structure des paquets NDN (Interest et Data)

Les deux paquets ont trois champs obligatoires, le premier sert à identifier le type de paquet soit un **Interest** ou un **Data**, et il est codé sur un octet de telle façon :

- Si ID paquet = A = $(01000001)_2$: le paquet est de type **Data**.
- Si ID paquet = B = $(01000010)_2$: le paquet est de type **Interest**.

Le champ Name des deux paquets définit le nom du paquet, il est codé sur 32 octets puisque la définition des noms dans NDN est sous forme d'URL peuvent contenir plusieurs caractères (**exemple** : /Youtube/video/exemple/vid.avi**).**

Le champ Nonce dans un paquet Interest sert à bien identifier un intérêt parmi d'autres d'une façon plus précise. Le dernier champ du paquet Data contient la donnée demandée.

3.4. Description de l'architecture implémentée

Nous avons conçu un système qui permet de gérer les mémoires **TCAM** d'un routeur de la future architecture d'internet qui est l'architecture **NDN**. Pour cela deux type de mémoire TCAM ont été implémentées à savoir **Content Store** et **Pending Interest Table**. Gérées par un algorithme spécifique. Nous expliquons dans ce qui suit le fonctionnement de notre algorithme implémenté et illustré par la (**Figure 3.18**).

Cet algorithme est en quelque sorte une machine d'états. Ce schéma permet de mieux expliquer l'implémentation de l'algorithme. Nous présentons les états et les signaux existants :

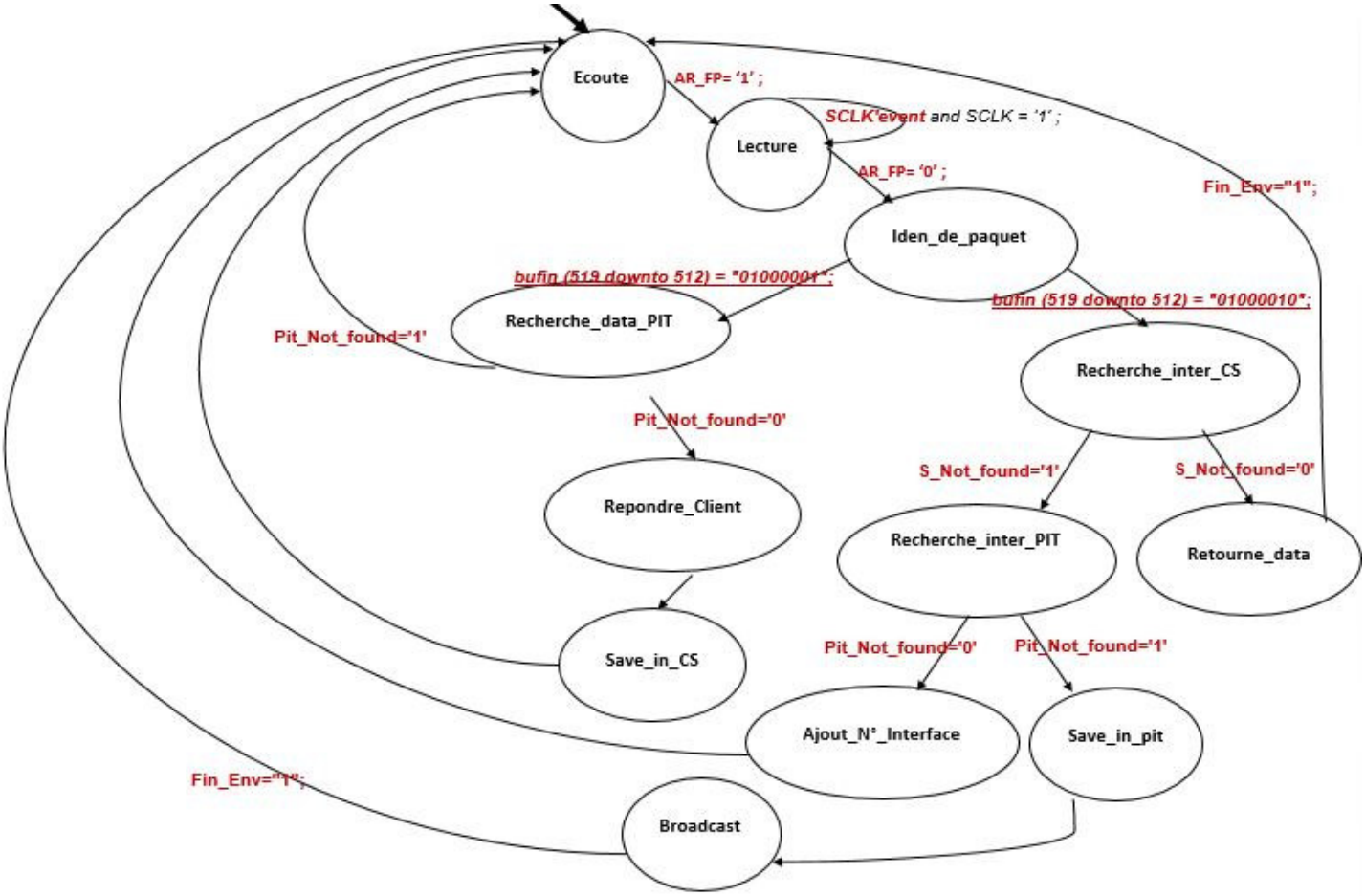

Figure 3.18: Le fonctionnement de l'algorithme implémenté

1- Dans un premier temps, l'algorithme est à l'état d'écoute de son environnement. Si un utilisateur désire émettre une donnée, il suffit de mettre la ligne **AR_FP = '1'** utilisée dans la communication **SPI** afin d'indiquer le début et la fin de la transmission ainsi le système transite vers une autre étape qui la lecture.

2- La lecture permet de lire bit par bit la donnée reçue. Dans cette étape l'information est reçue et stockée dans un registre (buffer) de réception pour des prochains traitements.

3- Après avoir reçu le paquet, il reste à l'analyser et identifier son type selon son premier octet (situé respectivement entre le bit **519** à **512**), afin de savoir quelle est la prochaine procédure. Si cet octet est un **A** alors le paquet est de type **Data** et si l'octet est un **B** le paquet est de type **Interest**.

4- ***Cas paquet Data*** *:* l'algorithme extrait le champ **Name** de ce paquet et le cherche dans la mémoire **PIT** dans l'état **Recherche_data_pit**, en cas d'échec de la recherche un signal d'erreur appelé **Pit_not_found** est mis à **'1'** logique. Ce qui explique qu'aucun **Interet** n'existe pour cette donnée car cette demande a été servie auparavant et qu'elle

a été supprimée par la suite, donc ce paquet sera ignoré afin d'éviter les redondances car le premier paquet **Data** qui arrive satisfait les demandes qui existent dans la PIT pour cette donnée et les prochains seront ignorés tant que la demande correspondante a été traitée. Si la recherche aboutit à une correspondance, le cas **Pit_not_found = '0'** alors la demande sera servie par ce paquet **Data** et ensuite il sera enregistré dans la **CS**.

5- ***Cas paquet Interest :*** une recherche dans la CS est effectuée dans l'état **Recherche_inter_cs**, en cas de **s_not_found = '0'** le paquet Data correspondant sera envoyé à l'interface d'entrée afin de satisfaire la demande, et tester vers la fin si **Fin_env = '1'** ce qui explique que l'envoie de ce paquet est terminé. Si aucune correspondance n'a été retrouvée le signal d'erreurs **s_not_found = '1'**, la prochaine étape est la recherche dans la PIT pour voir s'il y a déjà eu une réception de cet intérêt depuis d'autres interfaces dans l'état **Recherche_inter_PIT.**

6- Dans la **Recherche_inter_PIT,** à la recherche d'un intérêt dans la **PIT** il y a deux cas : cas de **PIT_not_found = '0'**, l'intérêt existe déjà dans la PIT, le numéro de l'interface qui a reçu l'intérêt est ajoutée à l'entrée. Le deuxième cas est lorsqu'il n'y a pas d'anciennes demandes, alors l'intérêt est enregistré et diffusé vers toutes les interfaces connectées dans l'état **Broadcast** afin de recevoir la donnée désirée.

3.5. Conclusion

Dans ce chapitre, nous avons présenté l'architecture de notre routeur NDN qu'on a pu créer et en décrivant les mémoires TCAM que nous avons implémenté, par la suite, nous avons démontré, que l'accès et la recherche dans ce type de mémoire se font très rapidement. Ainsi l'implémentation de ces mémoires pour les tables du protocole **NDN** (CS et PIT). Enfin nous concluons que les TCAM sont une solution optimale pour les problèmes (temps d'accès aux mémoires … etc) présents au niveau des routeurs NDN.

Chapitre 4 : Réalisation

Chapitre 4 : Réalisation

4.1. Introduction

Un système embarqué étant dédié à des tâches spécifiques, sa conception peut être optimisée pour réduire les coûts. Une bonne conception doit contenir juste assez de ressources matérielles pour répondre aux fonctionnalités requises par l'application.

L'évolution de la technologie permet la conception des systèmes embarqués complexes sur des dispositifs très puissants comportant des millions de transistors. Les derniers circuits **FPGA** permettent de réaliser des systèmes complets, qui répondent au mieux aux exigences des systèmes embarqués.

Le développement d'un système à base de FPGA est un processus complexe qui consiste en de nombreuses transformations complexes et algorithmes d'optimisation. Des outils logiciels sont nécessaires pour automatiser certaines tâches. Dans le cadre de notre projet, nous avons utilisé le package **Altera Quartus II Web Edition** pour la synthèse, l'implémentation et la programmation des périphériques, ainsi pour la simulation HDL. Dans ce chapitre, nous donnons un bref aperçu du périphérique FPGA et décrirons l'architecture de la carte de prototypage **ALTERA DE2**. Et pour le clôturer on décrira notre réalisation sur une telle architecture.

4.2. Généralités sur FPGA [10]

La technologie FPGA actuellement disponible permet de construire des systèmes qui sont configurables et/ou reconfigurables. L'idée commune à tous les circuits configurables est de proposer, non pas une fonction figée dont le champ d'application est le plus large possible, mais une structure adaptable. La reconfiguration consiste à modifier en cours de fonctionnement l'architecture matérielle, cela permet de concevoir des circuits intelligents qui peuvent s'auto adapter à leur environnement.

Les FPGA (field programmable gate array) sont inventés par la société Xilinx en 1985. Xilinx fut précurseur du domaine en lançant le premier circuit FPGA commercial, le XC2000. Ce composant avait une capacité maximum de 1500 portes logiques. La technologie utilisée était alors une technologie aluminium à 2 micromètres avec 2 niveaux de métallisation. Xilinx ne sera suivi qu'un peu plus tard, et jamais lâchée, par son plus sérieux concurrent Altera qui lança en 1992 la famille de FPGA FLEX 8000 dont la capacité maximum atteignait 15 000 portes logiques.

Les FPGA se situent entre les réseaux logiques programmables et les circuits logiques près diffusés. Les réseaux logiques programmables sont des composants qui ne nécessitent aucune étape technologique supplémentaire pour être personnalisés, ce sont des circuits standards, programmables par l'utilisateur grâce aux différents outils de développement et qui incluent un grand nombre de solutions basées sur les variantes de l'architecture des portes ET et OU. Les prés diffusés sont des circuits intégrés basés sur l'utilisation des réseaux de cellules dont les blocs ont été préalablement diffusés, il faut créer les connexions entre ces blocs. Les FPGA combinent donc à la fois la souplesse de la programmation des réseaux logiques programmables et les performances des circuits prés diffuses. En d'autres termes le FPGA permet d'avoir une architecture conçue sur mesure à haute densité dans un circuit électronique, avec la possibilité de modifier cette architecture quand des nouvelles applications apparaissent. Les langages HDL comme le VHDL ou le Verilog sont utilisés pour décrire les fonctionnalités qui seront implémentées sur le composant, puis la description matérielle est traduite dans un fichier de configuration pour le FPGA cible. La description matérielle peut être aussi utilisée pour la description d'un composant ASIC.

En 2000 et 2001, les deux concurrents Xilinx et Altera ont franchi une nouvelle étape au niveau de la densité d'intégration en proposant respectivement leurs circuits Virtex et Apex-II dont les capacités maximums avoisinaient les 4 millions de portes logiques équivalentes avec de plus l'introduction de larges bancs de mémoires embarquées. Aujourd'hui, les fréquences de fonctionnement de ces circuits sont de l'ordre de quelques centaines de Méga Hertz (ces dernières sont en réalité très dépendantes de l'application). À partir des années 2000, les capacités des FPGA ont permis d'offrir aux concepteurs une solution supplémentaire de réalisation pour une majorité d'applications. De plus, les outils de mise en œuvre des FPGA ont évolué et bien qu'encore pénalisants lors de la conception, ils permettent la réalisation rapide d'applications complexes.

4.2.1. L'architecture interne des FPGA [11]

La particularité d'un FPGA vient de sa reconfigurabilité. Les éléments logiques ainsi que le réseau d'interconnexion peuvent être reconfigurés dans le but de supporter une autre application. En effet, les FPGAs sont composés de blocs mémoire qui permettent de configurer et de figer la fonctionnalité du circuit. Il existe plusieurs types de FPGA (les **SRAM-based** FPGA, les **Flash-based** FPGA et les **Antifuse-based** FPGAs), classés en fonction de la manière dont ils sont programmés. La majorité des FPGAs utilisent des mémoires statiques et sont appelés **SRAM-based FPGA**.

Les circuits FPGA utilisent deux types de cellules de base, les cellules d'entrées/sorties appelées **IOB** et les cellules logiques appelées **CLB**. Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurable. Donc les trois blocs principaux **Figure 4.19** sont les blocs logiques configurables, les blocs d'entrées/sorties et les ressources de communications.

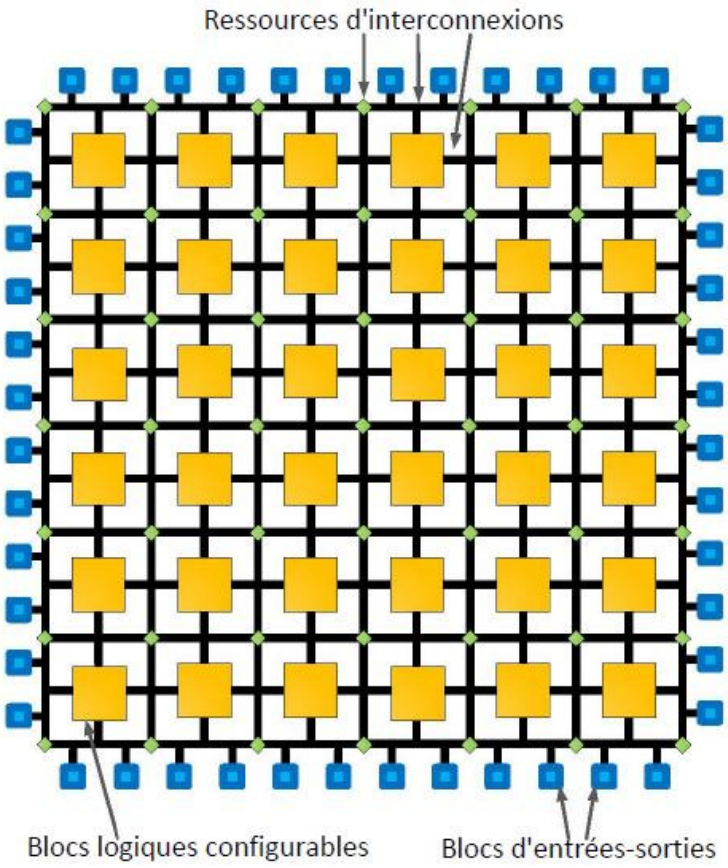

Figure 4.19: Architecture interne d'un FPGA

4.2.1.1. Les blocs principaux [11]

- **Les blocs logiques configurables (CLB)**

Les blocs logiques configurables sont les principaux éléments d'un FPGA. Ils peuvent avoir un ou plusieurs générateurs de fonctions réalisées avec des tables de correspondance (LUT - look-up tables) qui peuvent mettre en œuvre une logique arbitraire en fonction de leur configuration. Autour d'une LUT, il y a une logique d'interconnexion qui permet les liaisons à destination et vers la LUT. Elle est mise en œuvre à l'aide de portes logiques et de multiplexeurs. Pendant le processus de configuration d'un FPGA, les mémoires des LUT sont écrites pour y implémenter une fonction, et la logique qui l'entoure est configurée pour router

correctement les signaux afin de construire un système complexe. Il existe différents types de CLB en fonction du FPGA utilisé.

Les blocs d'entrées/sorties (IOB)

Les blocs d'entrée-sortie permettent l'interconnexion de la logique interne aux ports d'entrées et de sorties du FPGA. Les IOB ont leur propre mémoire de configuration, elle stocke les standards de tension et la direction des ports. Ces blocs sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé (haute impédance).

Les ressources d'interconnexion

Les ressources d'interconnexion au sein d'un **FPGA** permettent la connexion arbitraire des **CLB** et des **IOB**. Les connexions internes dans les circuits **FPGA** sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers **CLB**. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors **MOS** dont l'état est contrôlé par des cellules de mémoire vive ou **RAM**. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible. Pour parvenir à cet objectif, il existe trois sortes d'interconnexions selon la longueur et la destination des liaisons comme le montre la (**Figure 4.20**).

- **Les interconnexions directes :** Ces interconnexions permettent l'établissement des liaisons entre les CLB et les IOB. Il est possible aussi de connecter directement certaines entrées d'un CLB aux sorties d'un autre.
- **Les longues lignes :** Ce sont de longs segments métallisés parcourant toute la longueur et la largeur du FPGA, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible. De plus, ces longues lignes permettent d'éviter la multiplicité des points d'interconnexion.
- **Les matrices d'interconnexion :** Ce sont des aiguilleurs situés à chaque intersection. Leur rôle est de raccorder les longues lignes entre elles selon diverses configurations. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre CLB sur le FPGA pour assurer la communication des signaux. Pour éviter l'affaiblissement des

signaux traversant les longues lignes, des buffers sont implantés dans chaque matrice d'interconnexion.

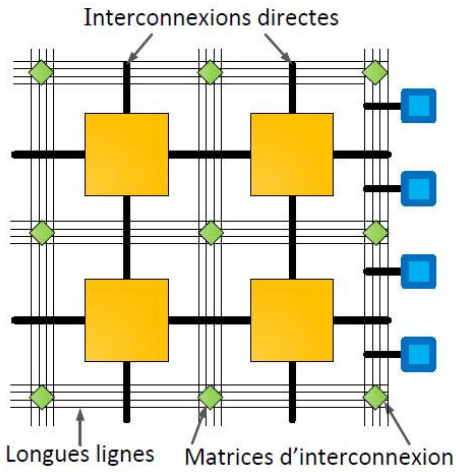

Figure 4.20: Interconnexion interne d'un FPGA

4.2.1.2.Les blocs à usages spécifique [11]

En plus des trois blocs décrits dans le paragraphe précédent, les FPGA possèdent souvent ce que l'on appelle des ressources additionnelles. La composition détaillée d'un circuit FPGA **Figure 4.21**, illustre la disposition de ces blocs additionnels sur un FPGA. Le fonctionnement et le placement de ces blocs diffèrent d'une référence de FPGA à une autre.

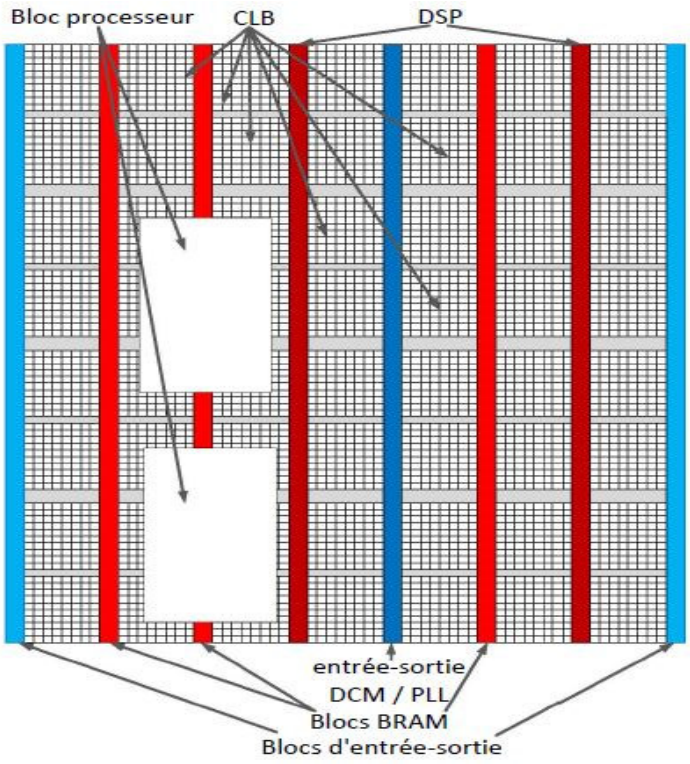

Figure 4.21: Ressources d'un circuit FPGA

- **Les blocs RAM (BRAM)**

Les blocs RAM sont des mémoires définies par l'utilisateur, embarquées sur le circuit intégré FPGA, elles servent à stocker des ensembles de données. En fonction de la catégorie de FPGA, la mémoire RAM embarquée est configurable en blocs de 16 ou de 32 kilo-octets. Les mémoires RAM sont de type double ports, elles permettent une écriture et une lecture indépendante sur chaque port avec une horloge différente Ceci est très utile, le composant peut produire (écrire) des données à une fréquence différente d'un autre composant qui consomme (lire) les données.

- **Digital Signal Processing (DSP)**

Les « Digital Signal Processing » sont des blocs qui permettent des conceptions plus complexes, qui peuvent consister soit en traitement numérique du signal ou seulement certains assortiments de multiplication, addition et soustraction. Comme pour les BRAM, il est possible de mettre en œuvre ces blocs grâce au CLB, mais il est plus efficace en termes de performances, et de consommation d'énergie d'intégrer plusieurs de ces composants au sein du FPGA. Un bloc DSP permet de réaliser un multiplicateur, un accumulateur, un additionneur, et des opérations logiques (AND, OR, NOT, et NAND) sur un bit. Il est possible de combiner les blocs DSP pour effectuer des opérations plus importantes, telles que l'addition avec virgule flottante simple, la soustraction, la multiplication, la division, et la racine carrée. Le nombre de blocs DSP est dépendant du dispositif.

- **Les processeurs embarqués**

Les processeurs embarqués enfouis sont l'un des ajouts les plus importants pour le FPGA. Beaucoup de conceptions nécessitent l'utilisation d'un processeur embarqué. Souvent, le choix d'un dispositif de FPGA avec un processeur embarqué permet de simplifier grandement le processus de conception tout en réduisant l'utilisation des ressources et la consommation d'énergie.

- **Le gestionnaire d'horloge numérique (DCM)**

Un gestionnaire d'horloge numérique permet d'avoir des périodes d'horloge différentes qui sont générées à partir d'une horloge de référence unique. La plupart des systèmes disposent

d'une horloge externe unique qui produit une fréquence d'horloge fixe. Cependant, il y a un certain nombre de raisons pour lesquelles un concepteur peut avoir besoin de fonction logique fonctionnant à des fréquences différentes. L'avantage d'utiliser un DCM est que les horloges générées auront moins de gigue.

4.2.2. Flux de développement [12]

Le flux de développement plus élaboré, qui comprend également la piste de vérification, est illustré à la **Figure 4.22** Pour faciliter la lecture, nous adoptons certains termes utilisés dans la documentation d'**Altera**. La piste de gauche du flux est le processus de raffinement et de programmation, dans lequel un système est transformé d'une description HDL textuelle abstraite en une configuration au niveau de la cellule de l'appareil, puis est téléchargé sur le périphérique FPGA. Le bon chemin est le processus de validation, qui vérifie si le système répond aux spécifications fonctionnelles et aux objectifs de performance. Les principales étapes du processus sont les suivantes :

1. Concevoir le système et dérivez le fichier VHDL (ou verilog). Nous devrons peut-être ajouter un fichier de contraintes distinct pour spécifier certaines contraintes d'implémentation.
2. Développer le **Testbench** en VHDL et effectuez une simulation RTL. Le terme RTL reflète le fait que le code VHDL est effectué au niveau du transfert de registre. La simulation vérifie si la description initiale du VHDL est conforme aux spécifications et aux fonctions.
3. Effectuer une analyse et une synthèse. Comme son nom l'indique, ce processus contient deux processus plus petits, l'analyse et la synthèse. Le processus d'analyse analyse la structure du fichier, vérifie la syntaxe des codes VHDL et procède à son élaboration, qui construit la hiérarchie de conception et établit la connectivité du signal. Le processus de synthèse transforme les constructions VHDL en composants génériques de niveau porte, puis les mappe aux éléments logiques et aux E / S du FPGA.
4. Effectuer le placement et le routage. Le processus de placement et de routage dérive l'agencement physique à l'intérieur de la puce FPGA. Il place les éléments logiques dans des emplacements physiques et détermine les itinéraires pour connecter différents signaux. Parfois appelé ajustement (fitting) dans la documentation Altera.

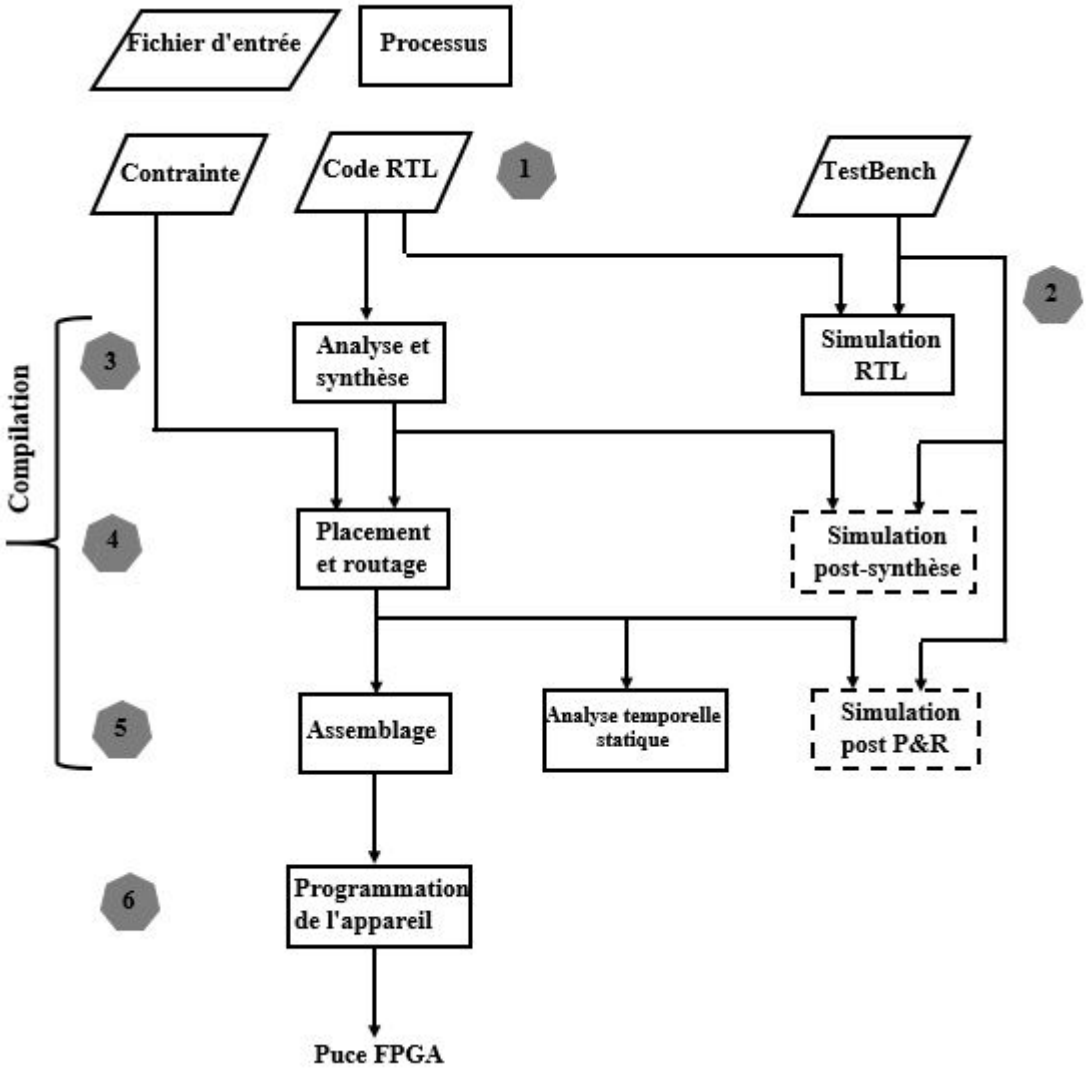

Figure 4.22: Flux de développement sur la carte FPGA

5. Générez le fichier de configuration. Dans cette étape, la sortie de l'installateur est convertie en un fichier "image de programmation" pour le périphérique cible désigné. C'est ce qu'on appelle l'assemblage dans la documentation d'**Altera**.

6. Programmez l'appareil. Dans ce processus, le fichier de configuration est téléchargé dans le périphérique cible. L'action est également appelée programmation d'un périphérique FPGA. Le circuit physique peut être vérifié en conséquence.

Les étapes 3, 4 et 5 combinées sont parfois appelées processus de compilation.

4.2.3. Présentation de la carte de développement (ALTERA DE2) [13]

Pour la mise en œuvre de notre projet, la carte de prototypage Altera **DE2** est utilisée et contient un périphérique FPGA de la famille Cyclone II d'Altera. Bien que les périphériques Cyclone II soient des périphériques FPGA d'entrée de gamme peu coûteux, ils disposent de toutes les fonctionnalités clés des périphériques avancés et prennent en charge l'utilisation d'un processeur Soft-Core.

La carte DE2 dispose de Cyclone II de type EP2C35F672C6, son circuit contient plus de 35 000 blocs logiques et de 767 broches d'entrées-sorties. La configuration du circuit (l'interconnexion entre tous les composants) est stockée en mémoire Flash intégrée de 4 MO. La carte est dotée de plusieurs périphériques externes permettant de tester diverses solutions.

4.2.3.1. Disposition et composant [13]

Une photo de la carte DE2 est illustrée à la **Figure 4.23**. Il décrit la disposition de la carte et indique l'emplacement des connecteurs et des composants clés.

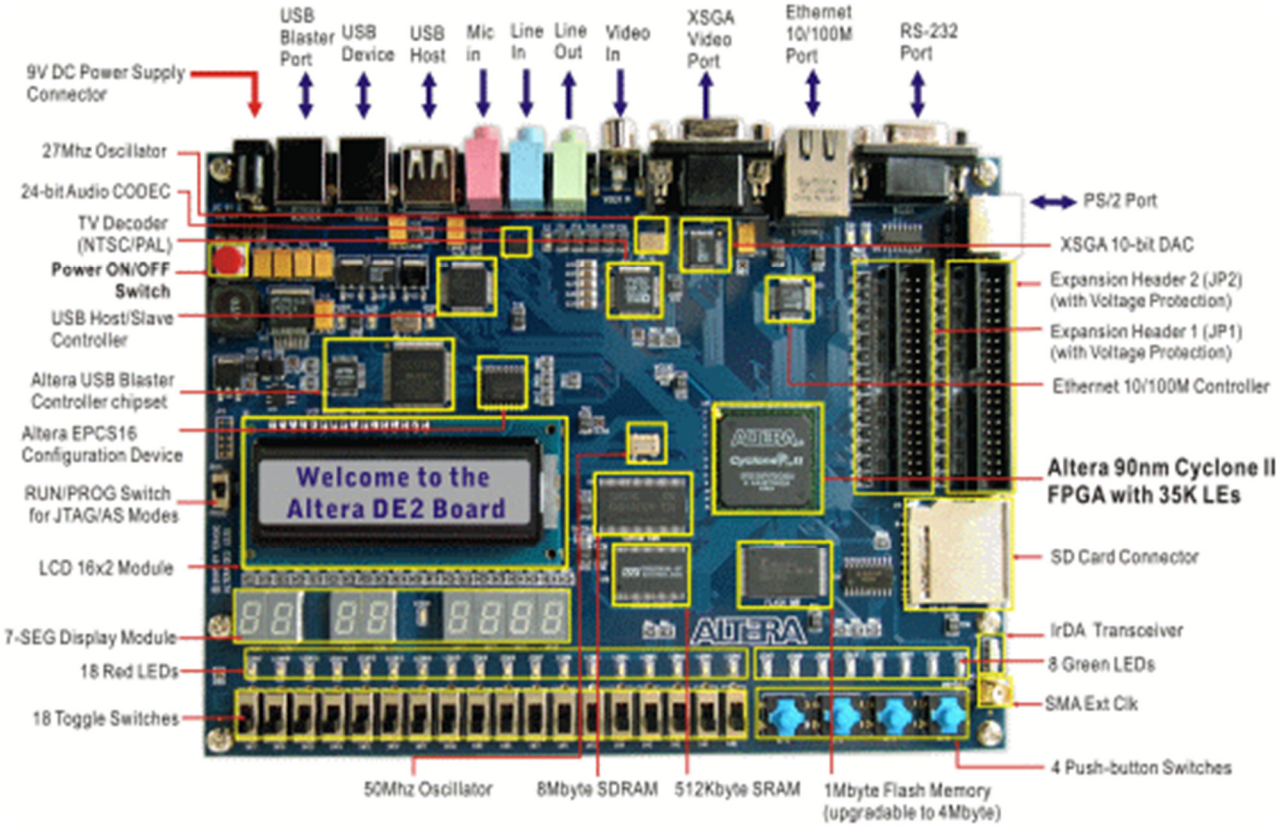

Figure 4.23: La carte de prototypage ALTERA DE2

La carte DE2 possède de nombreuses fonctionnalités qui permettent à l'utilisateur de mettre en œuvre une large gamme de circuits conçus, allant de circuits simples à divers projets multimédias. Le matériel suivant est fourni sur la carte DE2 :

- Périphérique FPGA Altera Cyclone II 2C35.
- Dispositif de configuration série Altera - EPCS16.
- Blaster USB (intégré) pour la programmation et le contrôle de l'API utilisateur, les modes de programmation JTAG et Active Serial (AS) sont pris en charge.
- SRAM de 512 Ko.
- SDRAM de 8 Mo.
- Mémoire Flash de 4 Mo.
- Lecteur de carte mémoire (SD CARD).
- 4 boutons poussoirs.
- 18 interrupteurs.
- 18 LED utilisateur rouges.
- 9 LED utilisateur vertes.
- Oscillateur 50 MHz et oscillateur 27 MHz pour les sources d'horloge.
- CODEC audio de qualité CD 24 bits avec prises : ligne d'entrée, ligne de sortie et entrée microphone.
- VGA DAC (triple DAC haute vitesse 10 bits) avec connecteur VGA-out.
- Décodeur TV (NTSC / PAL) et connecteur TV.
- Contrôleur Ethernet 10/100 avec connecteur.
- Contrôleur hôte / esclave USB avec connecteurs USB type A et type B.
- Émetteur-récepteur RS-232 et connecteur à 9 broches.
- Connecteur souris / clavier PS.
- Émetteur-récepteur IrDA.
- Deux têtes d'extension de 40 broches avec protection par diode.

En plus de ces fonctionnalités matérielles, la carte DE2 prend en charge les logiciels pour les interfaces E / S standard et une fonction de panneau de contrôle permettant d'accéder à divers composants.

4.2.3.2. Schéma fonctionnel de la carte DE2 [13]

La **Figure 4.24** donne le schéma fonctionnel de la carte DE2. Pour offrir une flexibilité maximale à l'utilisateur, toutes les connexions sont effectuées via le périphérique FPGA Cyclone II. Ainsi, l'utilisateur peut configurer le FPGA pour implémenter toute conception de système.

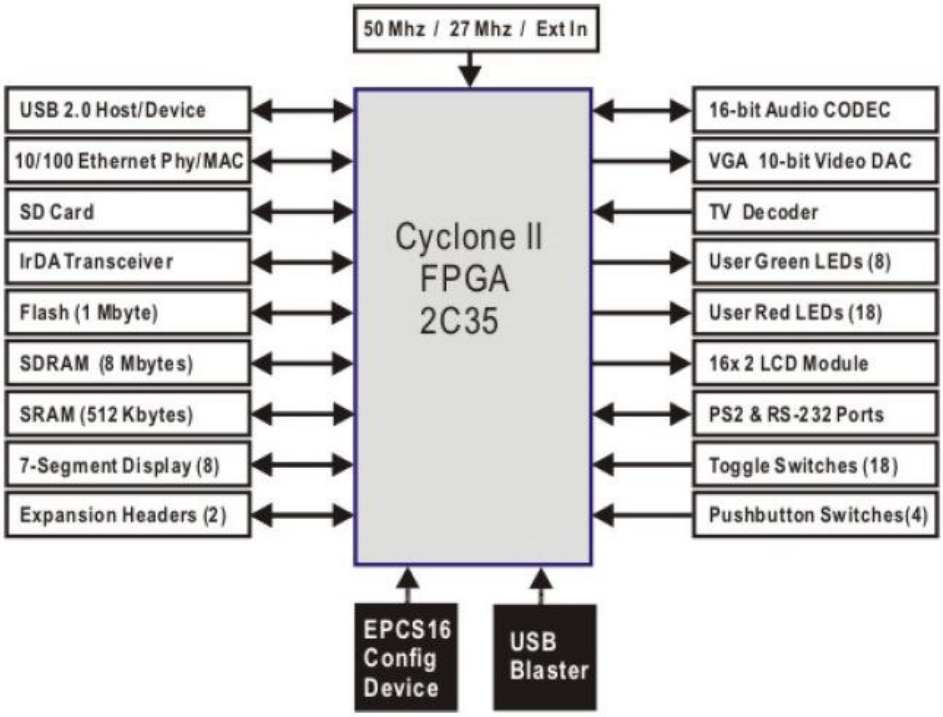

Figure 4.24: Schéma fonctionnel de la carte DE2

4.2.3.3.Vue d'ensemble de l'environnement de développement (Quartus II) [14]

Pour utiliser la carte DE2, l'utilisateur doit être familiarisé avec le logiciel Quartus II. Il existe trois versions de Quartus II basées sur la méthode d'entrée de conception utilisée, à savoir Verilog, VHDL ou entrée schématique.

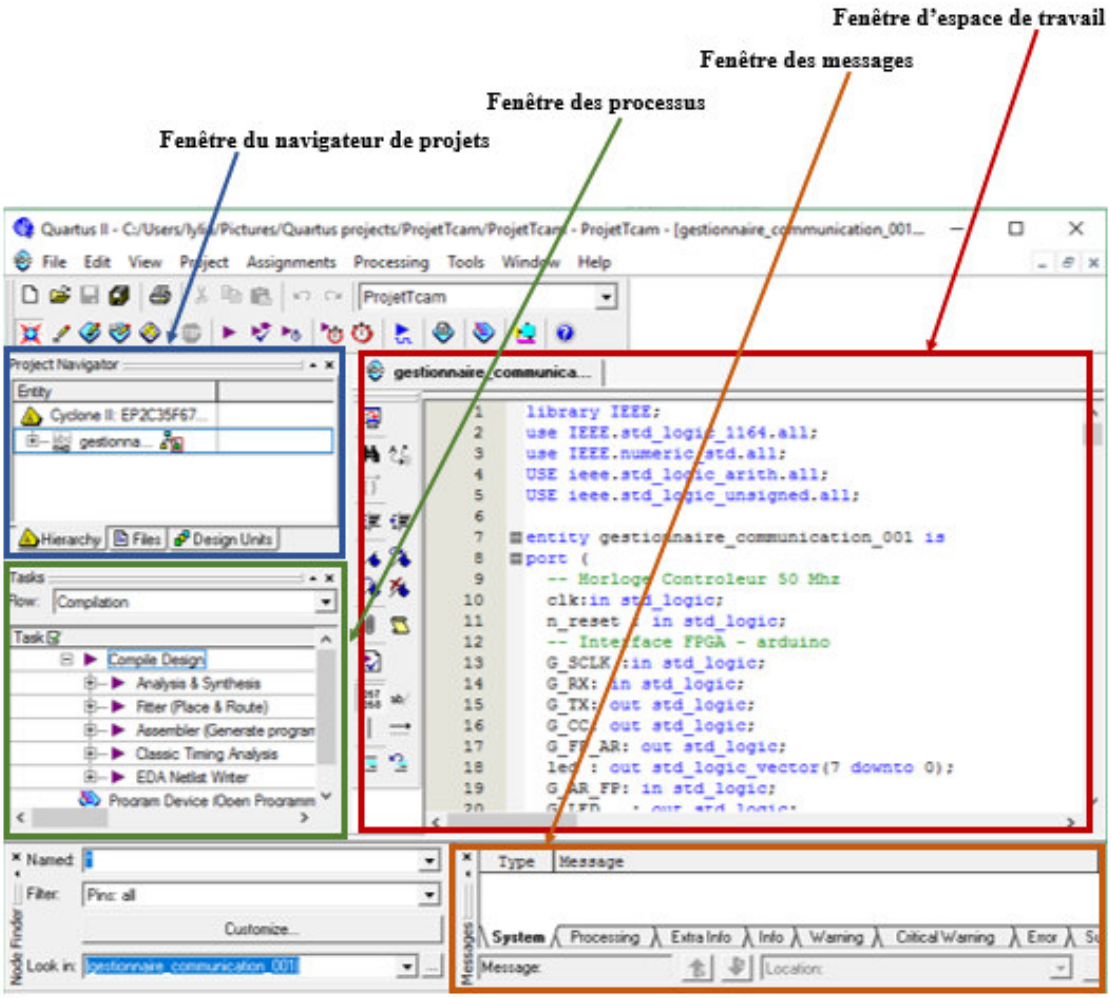

Figure 4.25: Interface graphique du logiciel Quartus II

Altera Quartus II contient des outils logiciels pour la partie de gauche. Il fournit une interface graphique permettant aux utilisateurs d'accéder aux outils et d'afficher les fichiers pertinents. Le logiciel Altera est mis à jour régulièrement. Certaines différences peuvent exister dans d'autres versions. La fenêtre graphique par défaut de Quartus II est illustrée à la (**Figure 4.25)**. Ses éléments de menu et les icônes d'action fréquemment utilisées sont affichés en haut. Le reste est divisé en quatre fenêtres plus petites :

- Fenêtre Project Navigator (en haut à gauche).
- Fenêtre Tâches (milieu gauche).
- Fenêtre Messages (bas).
- Zone de travail (en haut à droite).

4.3. Généralité sur Arduino [15]

Les cartes Arduino sont conçues pour réaliser des prototypes et des maquettes de cartes électroniques pour l'informatique embarquée. Ces cartes permettent un accès simple et peu couteux à l'informatique embarquée. De plus, elles sont entièrement libres de droit, autant sur l'aspect du code source (Open Source) que sur l'aspect matériel (Open Hardware). Ainsi, il est possible de refaire sa propre carte Arduino dans le but de l'améliorer ou d'enlever des fonctionnalités inutiles au projet.

Le langage Arduino se distingue des langages utilisés dans l'industrie de l'informatique embarquée de par sa simplicité. En effet, beaucoup de librairies et de fonctionnalités de base occulte certains aspects de la programmation de logiciel embarquée afin de gagner en simplicité. Cela en fait un langage parfait pour réaliser des prototypes ou des petites applications.

4.3.1. Caractéristiques techniques de l'Arduino UNO [15]

Un des modèles les plus répandu de carte Arduino est l'Arduino UNO (voir **Figure 4.26**). C'est la première version stable de carte Arduino. Elle possède toutes les fonctionnalités d'un microcontrôleur classique en plus de sa simplicité d'utilisation.

Elle utilise une puce ATmega328P **[1]** cadencée à **16Mhz**. Elle possède **32ko** de **mémoire flash** destinée à recevoir le programme, **2ko** de **SRAM** (mémoire vive) et **1 ko d'EEPROM** (mémoire morte destinée aux données). Elle offre **14 pins** (broches) d'entrée/sortie numérique (données acceptée 0 ou 1) **[2]** dont **6** pouvant générer des **PWM** (Pulse Width Modulation). Elle permet aussi de mesurer des grandeurs analogiques grâce à ces **6 entrées analogiques [3]**. Chaque broche est capable de délivré un courant de **40mA** pour une tension de **5V**.

Cette carte Arduino peut aussi s'alimenter et communiquer avec un ordinateur grâce à son port USB **[4]**. On peut aussi l'alimenter avec unes alimentions comprise en 7V et 12V grâce à sa connecteur *Power Jack* **[5].**

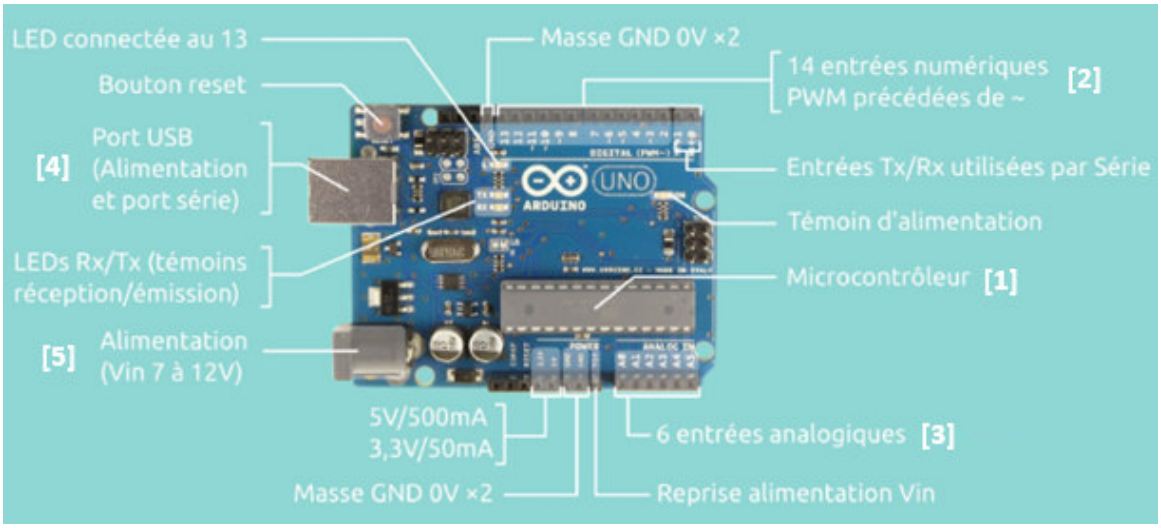

Figure 4.26: Caractéristiques de la carte Arduino UNO

4.3.2. IDE Arduino [15]

Un **IDE** (environnement de développement) libre et gratuit est distribué sur le site d'Arduino (compatible Windows, Linux et Mac). D'autres alternatives existent pour développer pour Arduino (extensions pour CodeBlocks, Visual Studio, Eclipse, XCode, etc).

L'interface de l'IDE Arduino est plutôt simple (voir **Figure 4.27**), il offre une interface minimale et épurée pour développer un programme sur les cartes Arduino. Il est doté d'un éditeur de code avec **coloration syntaxique [1]** et d'une **barre d'outils rapide [2].** Ce sont les deux éléments les plus importants de l'interface, c'est ceux que l'on utilise le plus souvent. On retrouve aussi une barre de menus **[3]** plus classique qui est utilisé pour accéder aux fonctions avancées de l'IDE. Enfin, une **console [4]** affichant les résultats de la compilation du code source, des opérations sur la carte, etc.

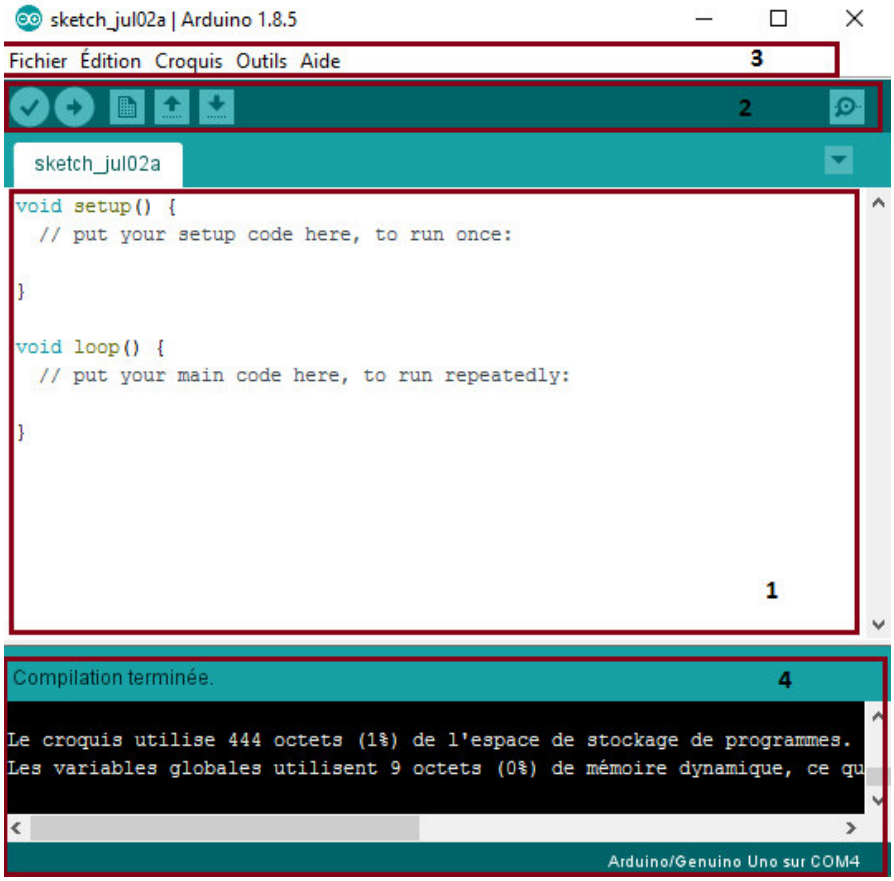

Figure 4.27: IDE Arduino

4.4. Circuit de gestion principal

Ce circuit est composé de deux composants principaux, l'un s'occupe de la gestion de la communication et de l'envoie des paquets qui nous permet d'assurer une connexion série de type SPI. Et le deuxième composant est le gestionnaire des deux TCAM (**CS_512x8** et **PIT_260x8**). **Figure 4.28** montre le schéma **RTL** après la synthèse du circuit gestion qui permet de gérer ces trois composants.

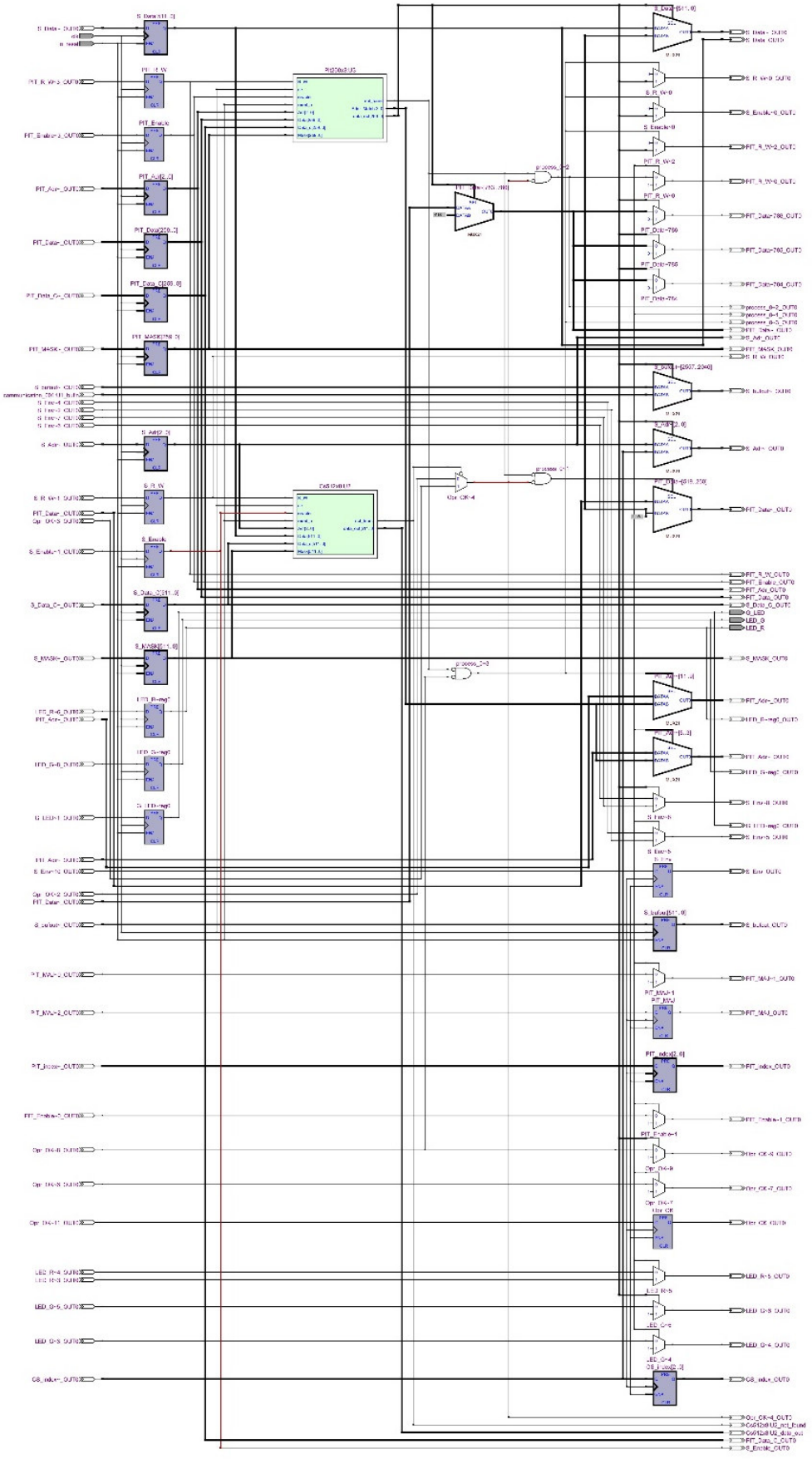

Figure 4.28: Schéma RTL du circuit de gestion principal

4.5. Simulations

4.5.1. La cellule TCAM

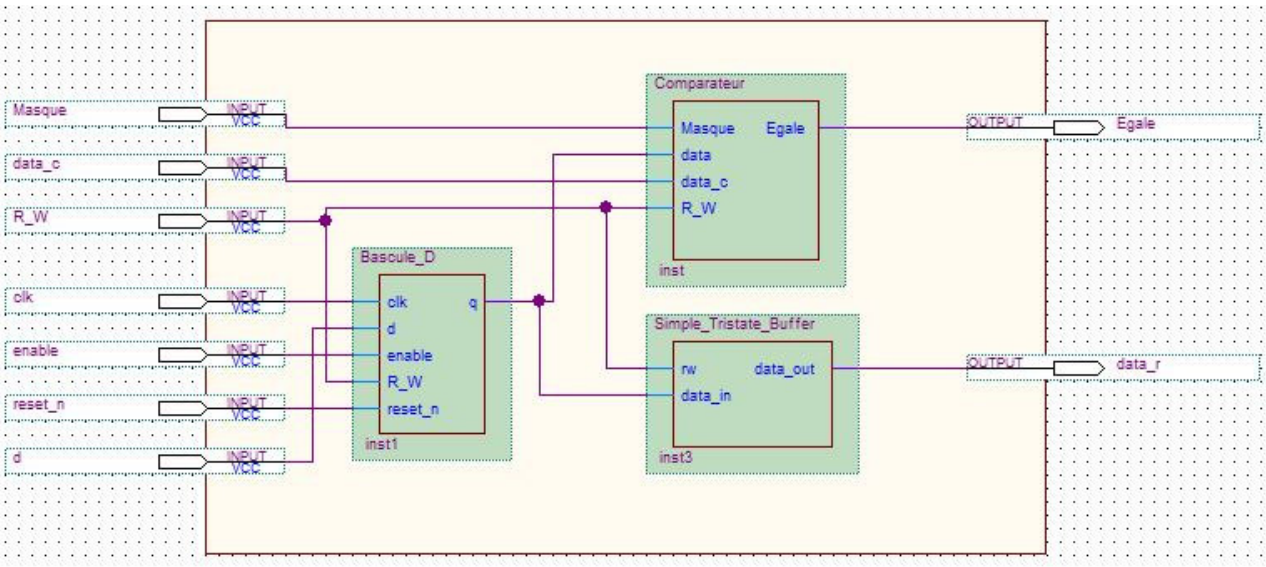

Figure 4.29: Le composant d'une cellule TCAM

La cellule TCAM stocke un bit reçu par l'entrée **d**, et le compare ensuite avec un bit reçu par l'entrée **data_c**. Cette cellule contient donc une **bascule D** reliée à un **comparateur** et en sortie elle définit le résultat de la comparaison dans la sortie **Egale** et retourne la donnée par **data_r.**

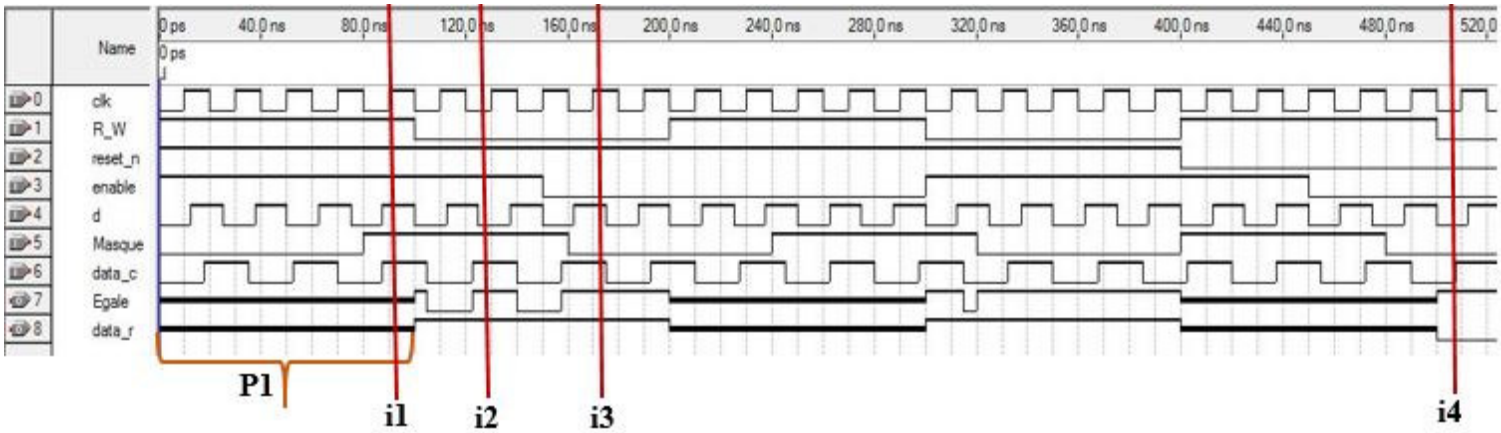

Figure 4.30: Schéma de simulation de la cellule TCAM

Analyse :

D'après la (**Figure 4.30**), la cellule TCAM contient plusieurs entrées qui régulent son fonctionnement, ces entrées sont les suivantes **:**

L'entrée R_W : lorsqu'elle est à **'1'**, elle permet l'écriture dans la cellule et quand elle passe à **'0'** elle bloque l'écriture pour effectuer la lecture de la dernière valeur écrite.

L'entrée reset_n : quand **reset_n = '1'** la cellule fonctionne normalement, mais lorsqu'elle passe à **'0'** le processus de réinitialisation se déclenche et met le contenu de la cellule à **'0'.**

L'entrée enable : celle-ci permet **la sélection** de la cellule TCAM précise, **enable = '1'** signifie que cette cellule est sélectionnée et quand elle est à **'0'** elle n'est pas sélectionnée donc elle mémorise son état précédent.

L'entrée Masque : lorsqu'elle est à **'1'** la comparaison entre **d** et **data_c** est effectuée et nous retourne son résultat par la sortie **egale,** et quand elle est à **'0'** y a pas de comparaison donc on met la sortie **egale = '1'** par défaut.

- **La période P1 :** dans cette période **R_W = '1'**, alors la cellule effectue une écriture à chaque top d'horloge (le fonctionnement habituel de la **bascule D**) et les opérations de comparaison, de lecture sont exclus et cela nous met les deux sorties **egale** et **data_r** en état **haute impédance (Z).**

- **A l'instant i1 :** les trois entrées **R_W, reset_n** et **enable** sont à **'1'**, et la dernière valeur écrite dans la cellule est un **'1'** comme le montre cet instant.

- **A l'instant i2 : R_W = '0'** une lecture est effectuée, de plus l'entrée **masque** est à **'1' alors** on compare la donnée présente dans la cellule qui est la dernière valeur **'1'** avec **data_c = '1'** et elles sont égaux, alors la sortie **egale** est mise à **'1'** et la donnée **'1'** est retournée par la sortie **data_r** comme on le constate.
- **A l'instant i3 :** l'entrée **enable = '0'** alors la cellule mémorise l'état précédant qui est à **'1'** et ignore toutes autres opérations sur les données, et **egale = '1'** car l'entrée **masque = '0'**.
- **A l'instant i4 :** l'entrée **reset_n = '0'** ce qui permet une réinitialisation ou la mise à **'0'** du contenu de la cellule qui est retournée par **data_r = '0'**.

- D'après l'analyse, les résultats obtenus sont corrects et correspondent bien au fonctionnement des mémoires TCAM. Alors notre TCAM est validée et sera utilisé prochainement pour la création d'un registre.

4.5.2. Le registre du content store

Figure 4.31: Le composant du registre de 64 octets qui constitue le content store

Un registre est un regroupement de cellules TCAM, donc pour faire un registre de 512 bits (64 octets) on utilise 512 cellules TCAM. Afin de simplifier les schémas on a opté de créer ce registre de 64 octet (byte).

Le schéma suivant (**Figure 4.32**) est le circuit interne du composant du registre de 64 octets comportant huit (8) registres de 64 bits (64 cellules TCAM).

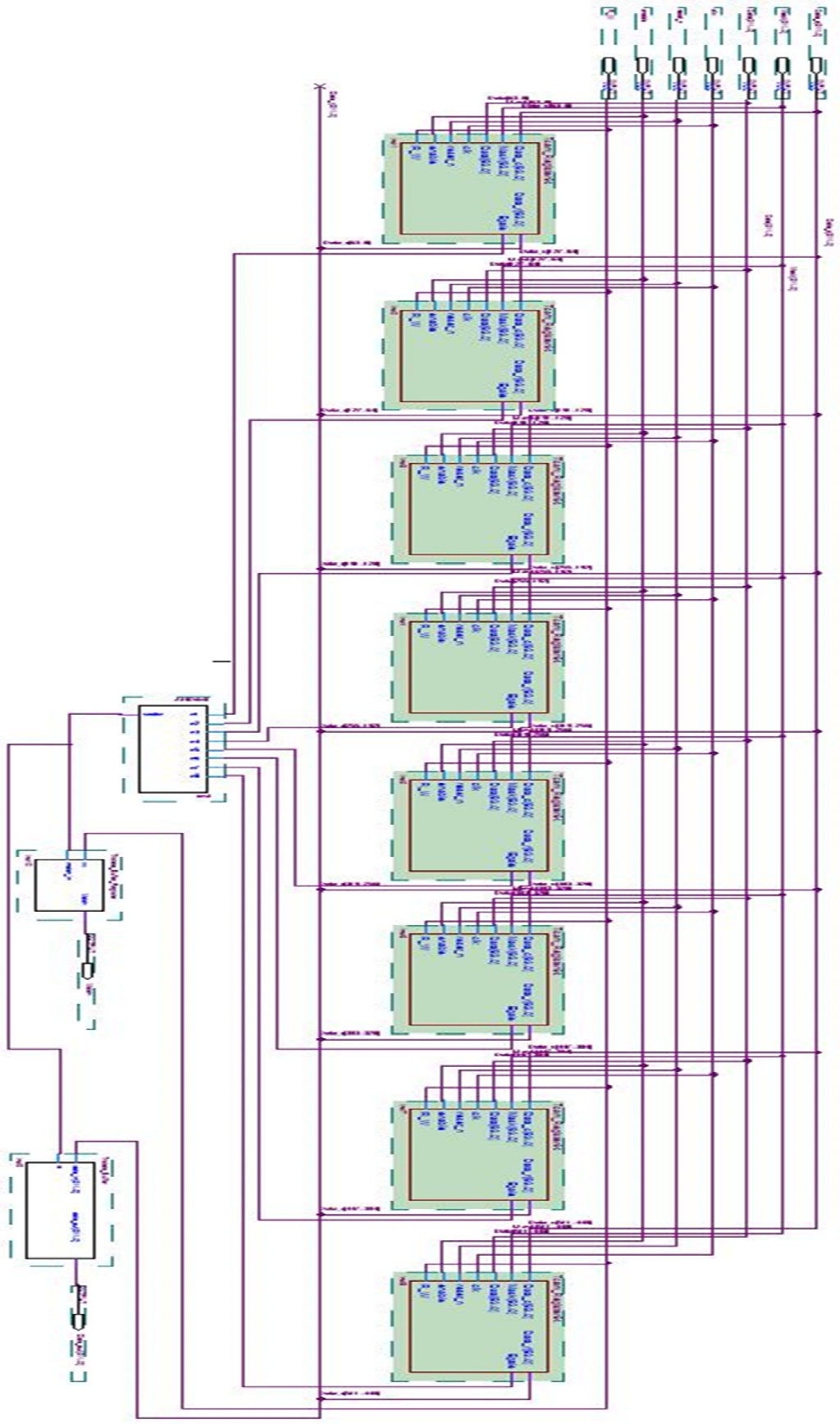

Figure 4.32: Schéma interne et complet du registre 64 octets

Pour la simulation de notre registre on a utilisé des valeurs en **décimal**, comme suit :

- **Cas reset_n = '1' :**

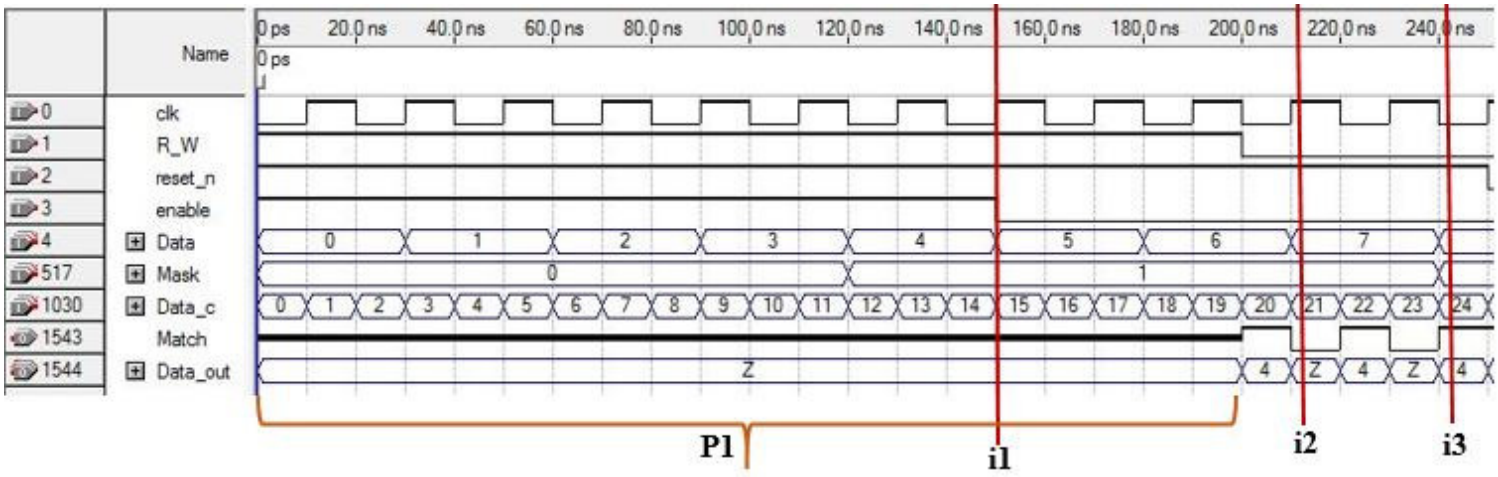

Figure 4.33: Schéma de simulation du registre TCAM quand reset_n = '1'

Analyse :

- **La période P1 :** l'entrée **R_W = '1'**, le registre est en mode d'écriture donc y a pas de comparaison et la ligne match est en haute impédance, ce qui met la sortie **Data_out** elle-même en haute impédance.

- **A l'instant i1 :** l'entrée **enable** passe à **'0'** ce que signifie que le registre ne fait pas d'écriture à cet instant et qu'il mémorise la dernière valeur qui est$('\mathbf{4}')_{\mathbf{10}}$.

- **A l'instant i2 :** les deux entrées **enable** et **R_W** sont à **'0'**, alors c'est le mode lecture qui est activé et le comparateur compare la valeur **Data_c** = $('\mathbf{21}')_{\mathbf{10}}$ avec la valeur du registre qui est $('\mathbf{4}')_{\mathbf{10}}$ comme suit :

$$\mathbf{Data} = ('4')_{10} = (\underbrace{'0 \ldots\ldots\ldots\ldots}_{509\ \text{zéros}}\mathbf{100}')_2$$

$$\mathbf{Mask} = ('1')_{10} = (\underbrace{'0 \ldots\ldots\ldots\ldots}_{511\ \text{zéros}}\mathbf{01}')_2$$

$$\mathbf{Data\_c} = ('21')_{10} = (\underbrace{'0 \ldots\ldots\ldots\ldots}_{507\ \text{zéros}}\mathbf{010101}')_2$$

En comparant les 1er bits du poids faible de $('\mathbf{4}')_{\mathbf{10}}$et de $('\mathbf{21}')_{\mathbf{10}}$ qui sont différents, donc le **Match = '0'** pour informer qu'il n y 'a pas de correspondance entre les deux données et aucune donnée n'est retournée dans **Data_out** cela est définie par la haute impédance (**Z**).

❖ **A l'instant i3 :** les deux entrées **enable** et **R_W** sont à '0', alors c'est le mode lecture et comparant la valeur Data_c = $('24')_{10}$ avec la valeur du registre qui est $('4')_{10}$ comme suit :

$$\mathbf{Data} = ('4')_{10} = ('\underbrace{0 \ldots\ldots\ldots\ldots}_{509\ \text{zéros}} 100')_2$$

$$\mathbf{Mask} = ('2')_{10} = ('\underbrace{0 \ldots\ldots\ldots\ldots}_{510\ \text{zéros}} 010')_2$$

$$\mathbf{Data\_c} = ('24')_{10} = ('\underbrace{0 \ldots\ldots\ldots\ldots}_{507\ \text{zéros}} 11000')_2$$

Les 2éme bits du poids faible de $('4')_{10}$ et de $('24')_{10}$ sont égaux, donc le **Match = '1'** qui signifie qu'il y a une correspondance, et la donnée $('4')_{10}$ est retournée par la sortie **Data_out.**

- **Cas reset_n = '0' :**

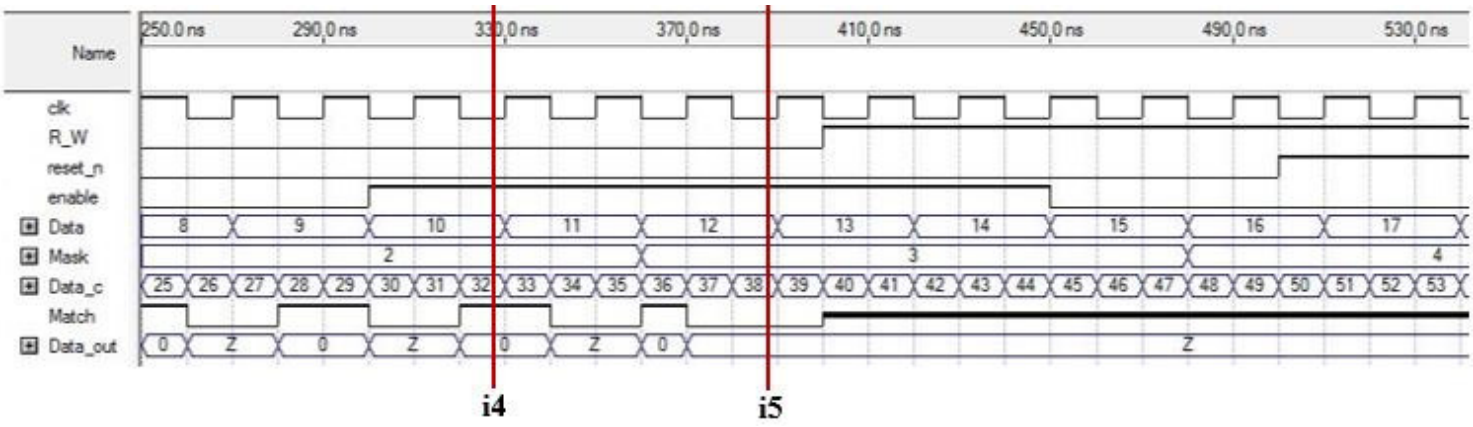

Figure 4.34: Schéma de simulation du registre TCAM quand reset_n = '0'

❖ **A l'instant i4 :** tant que l'entrée **reset_n = '0'** alors le registre se réinitialise et à présent le contenu du registre **Data**=$('0')_{10}$**.** Dans cet exemple le registre compare **Data_c** = $('32')_{10}$ comme suit :

$$\textbf{Data} = (\text{'0'})_{10} = (\underbrace{\text{'0} \dots \dots \dots \dots \text{000}}_{\text{512 zéros}}\text{'})_2$$

$$\textbf{Mask} = (\text{'2'})_{10} = (\underbrace{\text{'0} \dots \dots \dots \dots \text{010}}_{\text{510 zéros}}\text{'})_2$$

$$\textbf{Data\_c} = (\text{'32'})_{10} = (\underbrace{\text{'0} \dots \dots \dots \dots \text{0 100000}}_{\text{506 zéros}}\text{'})_2$$

Puisque le Mask = '2' alors le comparateur compare les 2éme bits du poids faible et qui est '0' pour les deux donnée **Data** = $(\text{'0'})_{10}$ et **Data_c** = $(\text{'32'})_{10}$. Donc on a en sortie le **Match** qui est à **'1'** et la donnée du registre qui est $(\text{'0'})_{10}$ est retournée.

- **A l'instant i5 :** l'entrée **Data** = $(\text{'38'})_{10}$ mais puisque **reset_n = '0'** alors la donnée sauvegardée est **réinitialisée** à **'0',** et elle est comparée avec **Data_c** comme suit :

$$\textbf{Data} = (\text{'0'})_{10} = (\underbrace{\text{'0} \dots \dots \dots \dots \text{0 00}}_{\text{512 zéros}}\text{'})_2$$

$$\textbf{Mask} = (\text{'3'})_{10} = (\underbrace{\text{'0} \dots \dots \dots \dots \text{011}}_{\text{510 zéros}}\text{'})_2$$

$$\textbf{Data\_c} = (\text{'38'})_{10} = (\underbrace{\text{'0} \dots \dots \dots \dots \text{0 100110}}_{\text{506 zéros}}\text{'})_2$$

La sortie **Match = '0'** car les 2éme et les 3éme bits de $(\text{'0'})_{10}$ et $(\text{'38'})_{10}$ sont différents, donc aucune donnée n'est retournée dans **Data_r** et la met en **haute impédance (Z)**.

- Les résultats obtenus sont conformes et d'où la validation de notre composant registre, qu'on réutilisera pour la TCAM finale qui est le Content Store.

4.5.3. La mémoire Content Store

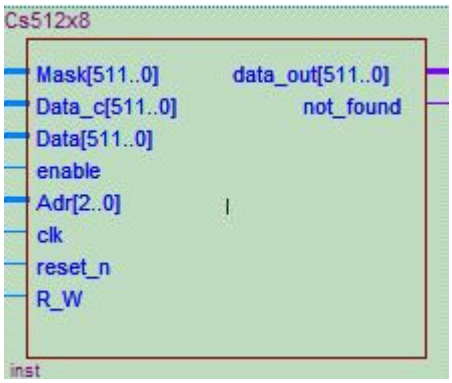

Figure 4.35: Le composant de la TCAM « Content Store ».

Notre TCAM « **Content Store** » contient huit lignes donc huit registres de 64 octets vu précédemment (**voir la Figure 4.35**), chacun de ces registres est sélectionné lors d'écriture à l'aide de l'entrée **Adr** comme le montre la **Figure 4.36**.

La recherche dans la TCAM se fais lorsque l'entrée **R_W = '0'**, l'entrée **Mask** définie les bits exactes qui doivent être comparés. En parcourant toute la TCAM cette dernière nous retourne le résultat de sa recherche par la sortie **data_out** et ainsi elle indique si la donnée est retrouvée ou non par l'entrée **not_found** qui peut être considérée comme erreur de recherche (aucune donnée n'est retrouvée) dans le cas ou **not_found = '1'.**

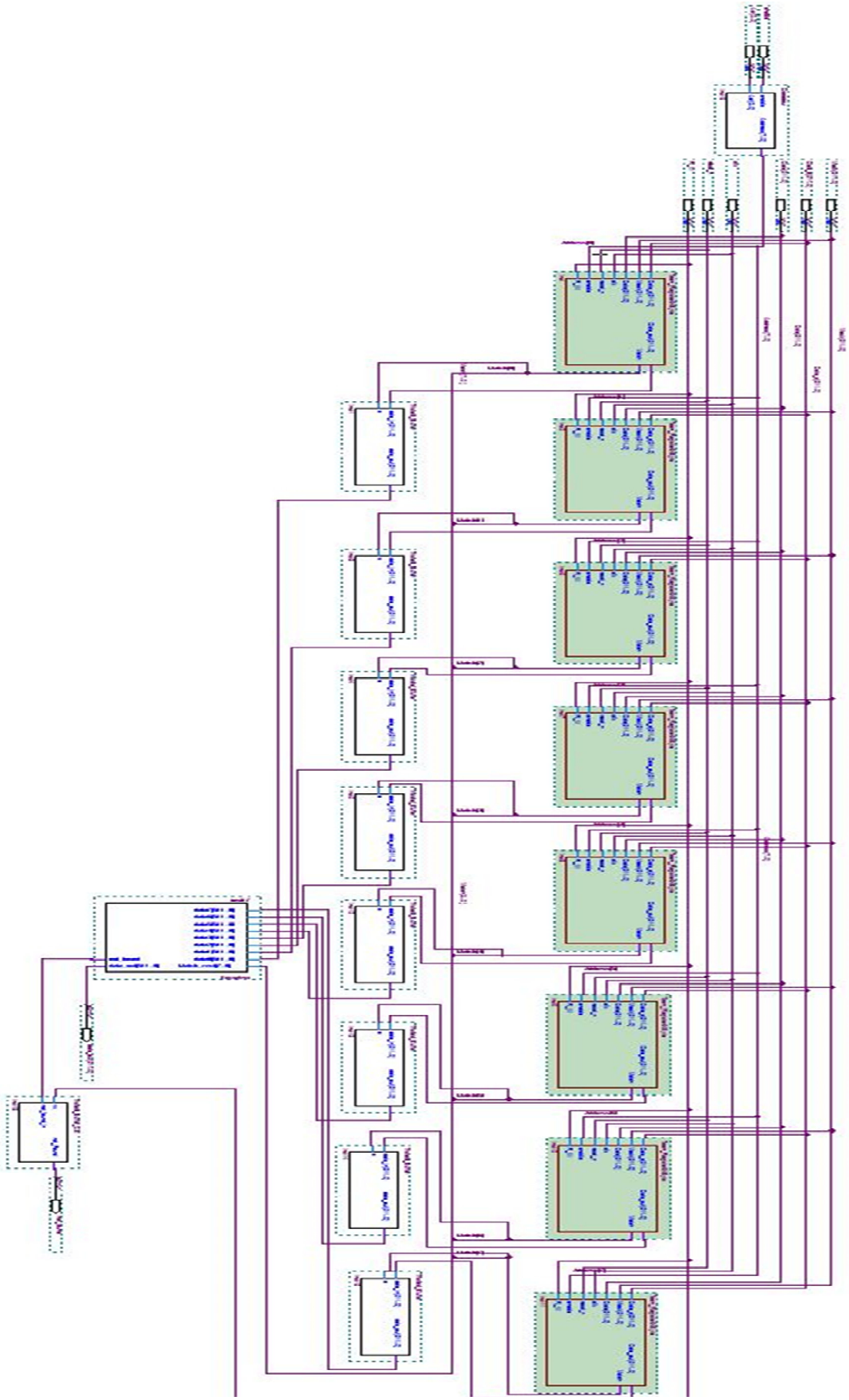

Figure 4.36: Schéma interne et complet de la TCAM « Content store »

- **Cas reset_n = '1' :**

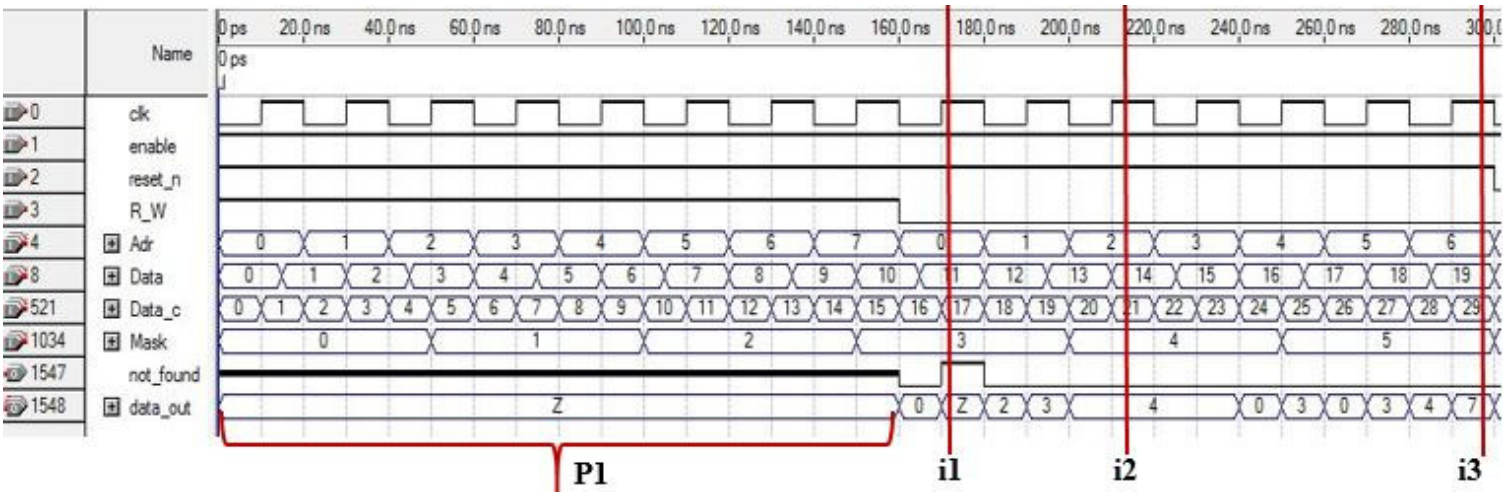

Figure 4.37: Schéma de simulation de la TCAM quand reset_n = '1'

Analyse :

- **La période P1 :** durant cette période la mémoire TCAM écrit dans ces cellules à chaque top de **clk** puisque l'entrée **R_W = '1'**, c'est pour cela qu'on obtient une haute impédance en sortie par **data_out**. La **TCAM** sera comme suit vers la fin de la période d'écriture :

Numéro de la ligne TCAM (Adr)	La donnée enregistrée (Data) (512 bits)
0	0= **0……00000**
1	2= **0……00010**
2	3= **0……00011**
3	4= **0……00100**
4	6= **0……00110**
5	7= **0……00111**
6	8= **0……01000**
7	10=**0……01010**

Tableau 3.1: Résultats de simulation de la TCAM - Content Store

L'écriture est terminée, et l'entrée R_W passe à '0' afin d'effectuer une lecture et comparaison des valeurs de la TCAM. Vérifions cela dans les instants suivants :

- **A l'instant i1 :**

Data_c = $('17')_{10} = ('0 \ldots\ldots\ldots\ldots 010001')_2$

Mask = $('3')_{10} = ('0 \ldots\ldots\ldots\ldots 011')_2$

La recherche dans cette TCAM se base alors sur les 2 bits du poids faible en parcourant toute la mémoire. La sortie **not_found = '1'** puisque il n'y a aucune lignes ne correspond avec cette donnée dans les deux bits **'10'**. Ce qui met la sortie **data_out** en haute impédance.

❖ **A l'instant i2 :**

Data_c = $('21')_{10}$ = $('0 \ldots\ldots\ldots\ldots 010101')_2$

Mask = $('4')_{10}$ = $('0 \ldots\ldots\ldots\ldots 0100')_2$

La donnée à comparer « **Data_c** » correspond à une donnée « **Data** » de la **4éme ligne** de la TCAM en se basant sur le 3éme bit, la sortie **not_found = '0'** donc y a aucune erreur car la donnée est retrouvée. La sortie **data_out** retourne la donnée de la ligne correspondante qui est $('4')_{10}$.

❖ **A l'instant i3 :**

Data_c = $('29')_{10}$ = $('0 \ldots\ldots\ldots\ldots 011101')_2$

Mask = $('5')_{10}$ = $('0 \ldots\ldots\ldots\ldots 0101')_2$

En se basant sur le 1er et le 3éme bit du poids faible lors de la recherche, la TCAM retrouve une correspondance avec la **ligne 5** car la donnée stockée dans cette ligne est $('7')_{10}$ = $('0 \ldots\ldots\ldots\ldots 0111')_2$. Donc la sortie **not_found = '0'** et la donnée $('7')_{10}$ est retournée dans **data_out**.

Cas reset_n = '0' :

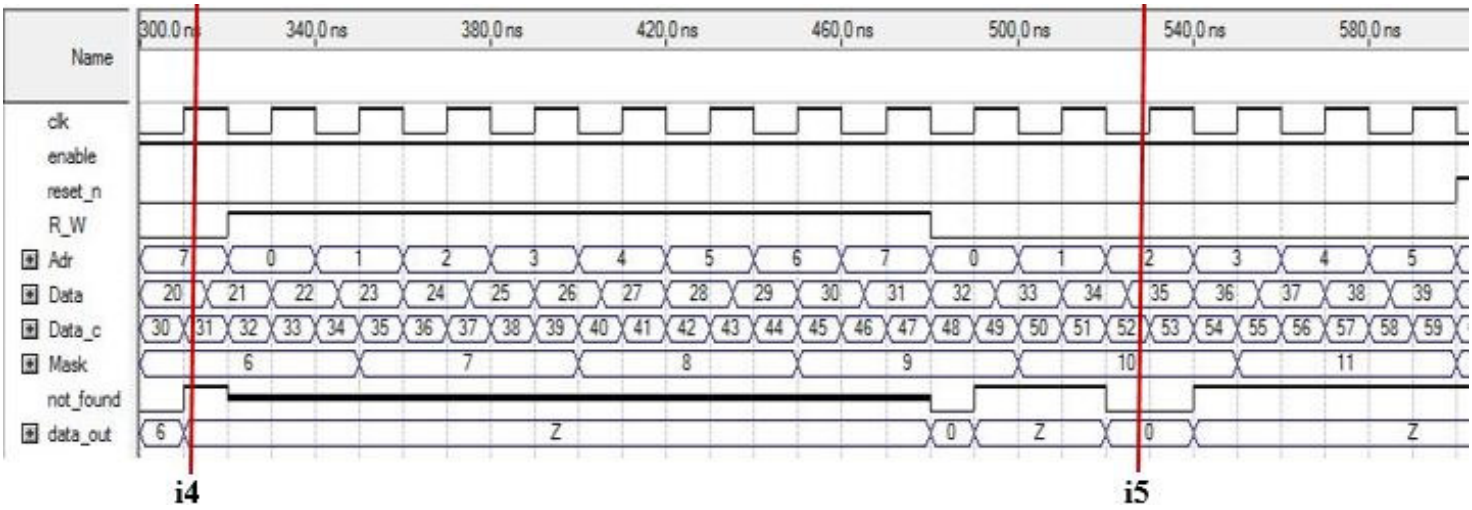

Figure 4.38: chéma de simulation de la TCAM à la réinitialisation (reset_n = '0')

L'entrée **reset_n = '0'** ce qui signifie que tous le contenu de la TCAM est mis à **'0'**, donc toutes les lignes contiennent à présent que des zéros et dans le cas où la donnée à comparer est

égale avec $('0')_{10}$ alors toutes les lignes de cette mémoire vont correspondre, mais la ligne qui sera prise est celle qui as l'adresse la plus basse c'est donc la **ligne 0**.

- **A l'instant i4 :**

Data_c = $('31')_{10}$ = $('0 \dots \dots \dots \dots 011111')_2$

Mask = $('6')_{10}$ = $('0 \dots \dots \dots \dots \dots 0110')_2$

Data = $('0')_{10}$ = $('0 \dots \dots \dots \dots 000000')_2$

On remarque que le 2éme et le 3éme bit de **Data_c** et de **Data** ne correspondent pas, et donc il y a une génération d'erreur par la sortie **not_found = '1'** et une haute impédance sur la sortie **data_out**.

- **A l'instant i5 :**

Data_c = $('52')_{10}$ = $('0 \dots \dots \dots \dots 011010 0')_2$

Mask = $('6')_{10}$ = $('0 \dots \dots \dots \dots \dots 01010')_2$

Data = $('0')_{10}$ = $('0 \dots \dots \dots \dots 000000')_2$

Les deux bits sont égaux dans **Data_c** et **Data**, alors toutes les lignes de la TCAM correspondent mais selon son fonctionnement elle prend la donnée de la ligne **d'adresse la plus** petite « **ligne 0** » et la retourne sur la sortie **data_out** = $('0')_{10}$**.**

- On conclut que de notre TCAM « **content store** » sont conformes au fonctionnement des TCAMs. De ce principe on peut validée notre mémoire TCAM et qui sera utilisé dans la suite du projet.

4.5.4. Le registre de la TCAM « Pending Interest Table »

Figure 4.39: Le composant du registre de 260 bits qui constitue la PIT

Le registre **PIT** contient **260 bits** dont **256 bits** pour la partie **Name** du paquet d'intérêt formé par des registres de **64 bits** en raison de simplification qui constituent le « **Name** »sur **32 octets**, et les **4 bits** restants sont destinés aux numéros d'interfaces de l'intérêt formé seulement par des cellules TCAM.

Les composants présentés précédemment forment un registre de 260 bits qui sera utilisé pour la création de la mémoire PIT.

Ce registre est semblable au registre de la mémoire **Content store** et sa simulation sera exactement la même. Pour cela on peut valider notre registre puisqu'il fonctionne correctement. Le schéma interne de ce registre est illustré par la **Figure 4.40**.

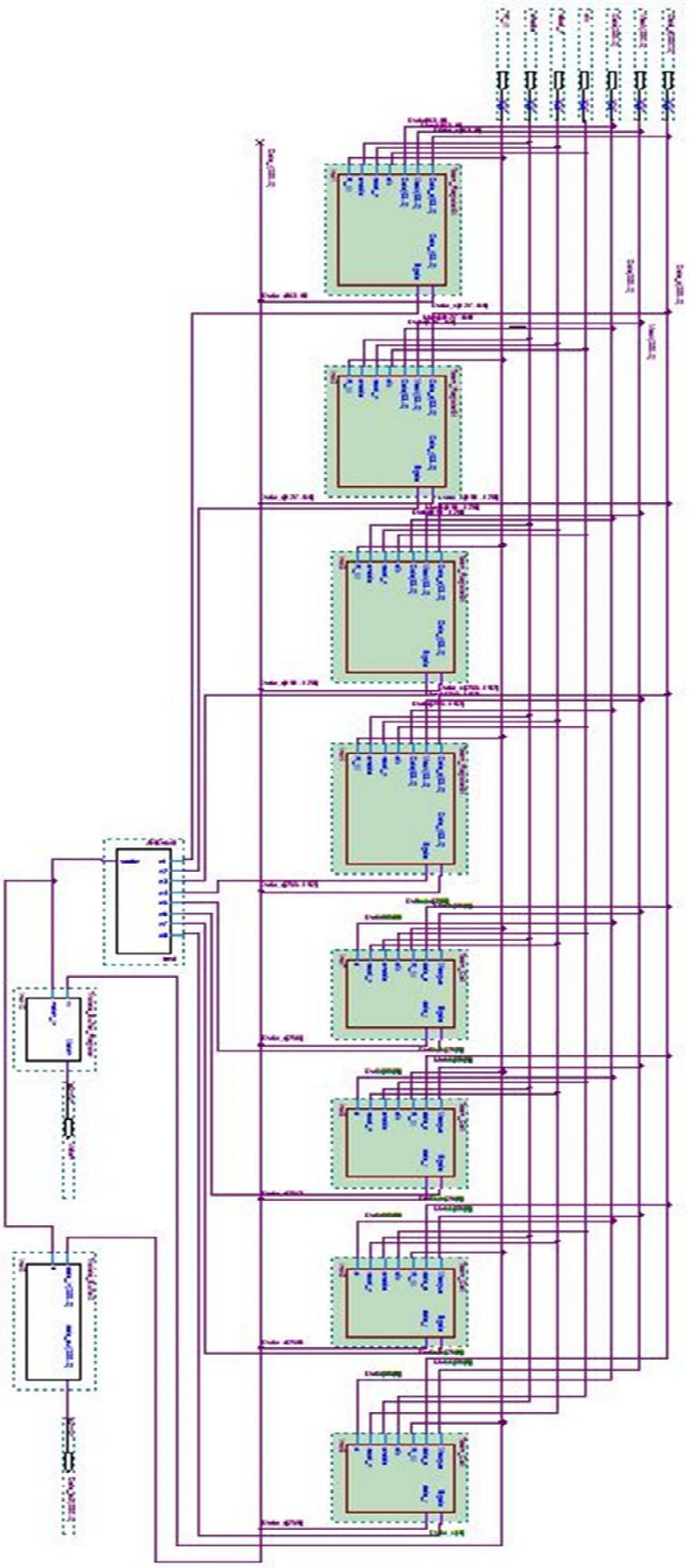

Figure 4.40: Schéma interne et complet du registre 260 bits de la PIT

4.5.5. La mémoire PIT

Figure 4.41: Le composant de la TCAM « Pending Interest Table »

Notre TCAM « **Pending Interest Table** » contient huit lignes donc huit registres de 260 bits vu précédemment (**voir la Figure 4.41**), chacun de ces registres est sélectionné lors d'écriture à l'aide de l'entrée **Adr** comme le montre la **Figure 4.41 et 4.42**.

La recherche dans la TCAM se fais lorsque l'entrée **R_W = '0'**, l'entrée **Mask** définie les bits exactes qui doivent être comparés dans le cas d'une PIT les bits a comparés sont les 256 bits qui définissent le champ Name (32 octets), ainsi les quatre derniers bits (**256 ... 259**) afin de spécifier le numéro de l'interface entrante respectivement (**1,2,…4**). Pour savoir si une demande est déjà été faite sur une interface spécifique il suffit de vérifier si le bit correspondant à l'interface est à **'1'.** Par exemple si **l'interface 1** a reçu un paquet, alors le bit **256** est à **'1'**, si c'était **l'interface 4** c'est le bit **269** qui est **à '1'**. En parcourant toute la TCAM cette dernière nous retourne le résultat de sa recherche par la sortie **data_out** qui est les numéros d'interfaces par lesquelles l'intérêt est reçu et ainsi elle l'indique par l'entrée **not_found** qui représente une erreur de recherche dans le cas où elle est à **'1'**.

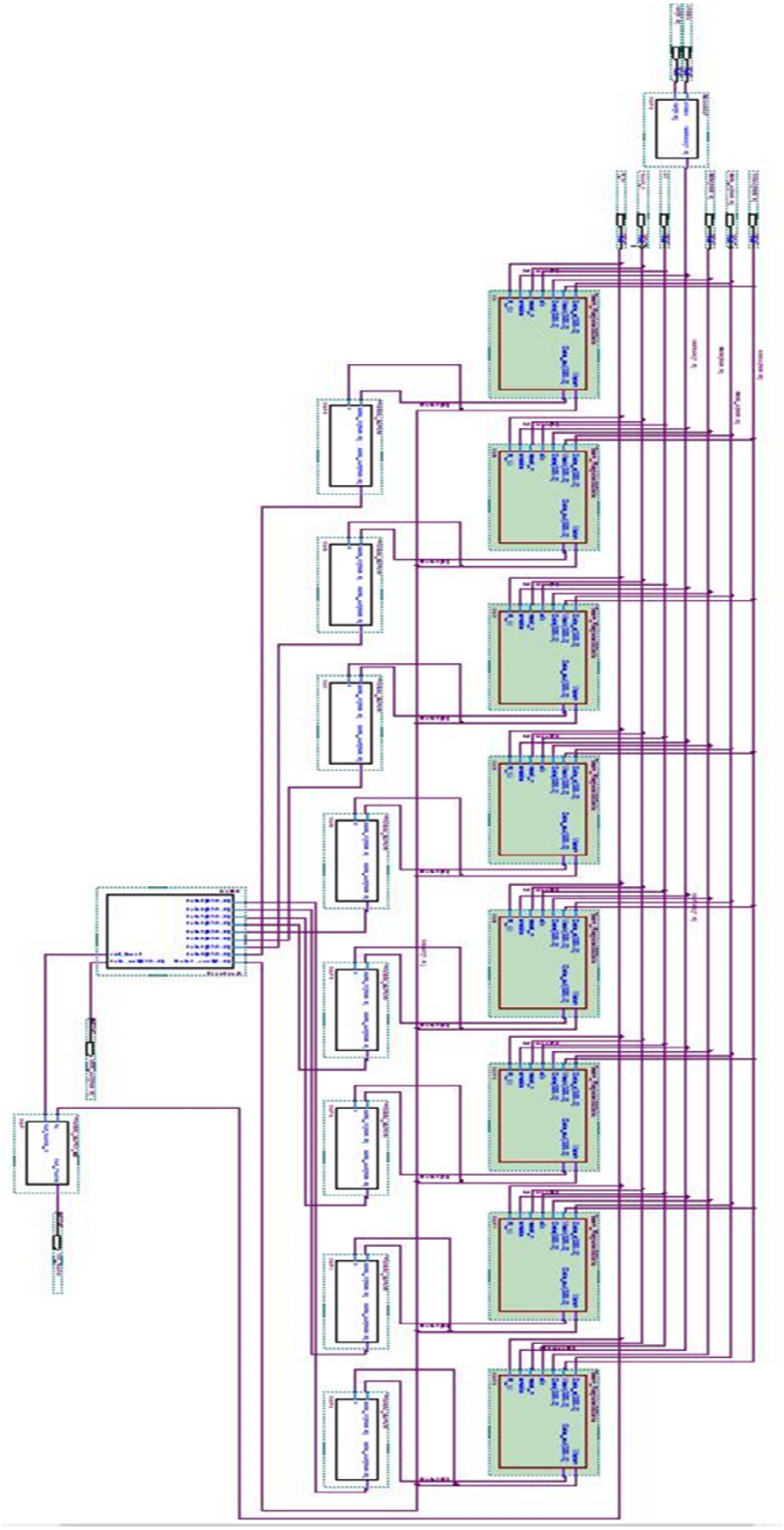

Figure 4.42: Schéma interne et complet de la TCAM « PIT »

Voici quelques schémas de simulation de cette PIT (**Figures 4.43** et **4.44**), afin de voir son fonctionnement et de la valider.

- **Cas 1 : reset_n = '1' :**

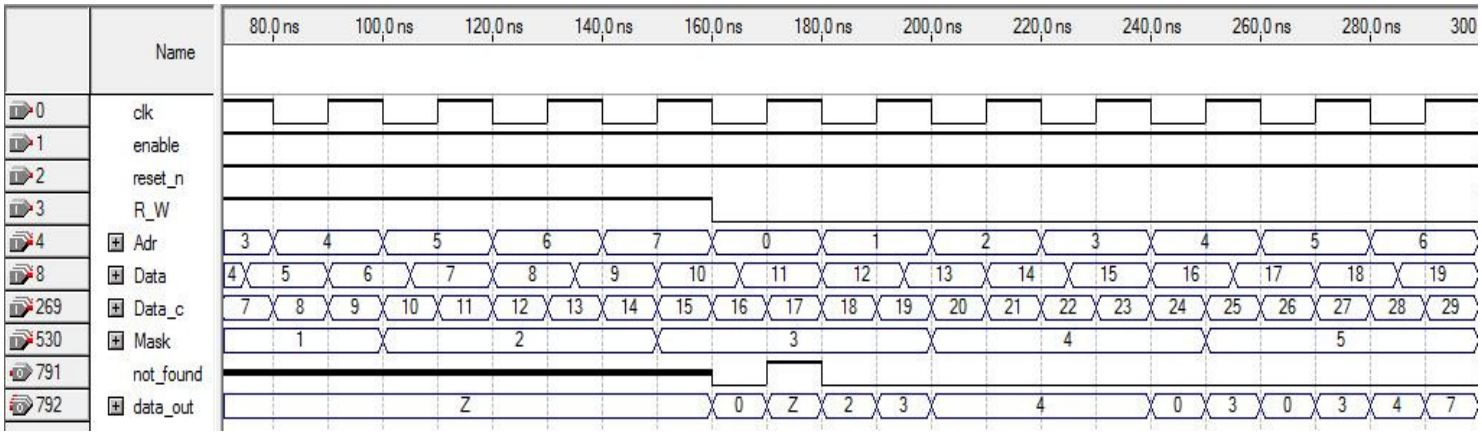

Figure 4.43: Schéma de simulation de la PIT « partie 1 »

Dans cette première partie de la simulation l'entrée **reset_n = '0'**, donc le fonctionnement est normal et on distingue deux autres cas : le cas de **R_W = '1'** qui spécifie une écriture. Pour cela les sorties sont en état d'haute impédance **'Z'**. par contre dans le cas **R_W = '0'** qui est une lecture, la mémoire compare la donnée reçue par l'entrée **Data_c**, et selon les bits spécifiés dans le **Mask** elle retourne le résultat de la comparaison par les deux sorties **not_found** et **Data_out**.

- **Cas 2 : reset_n = '0' :**

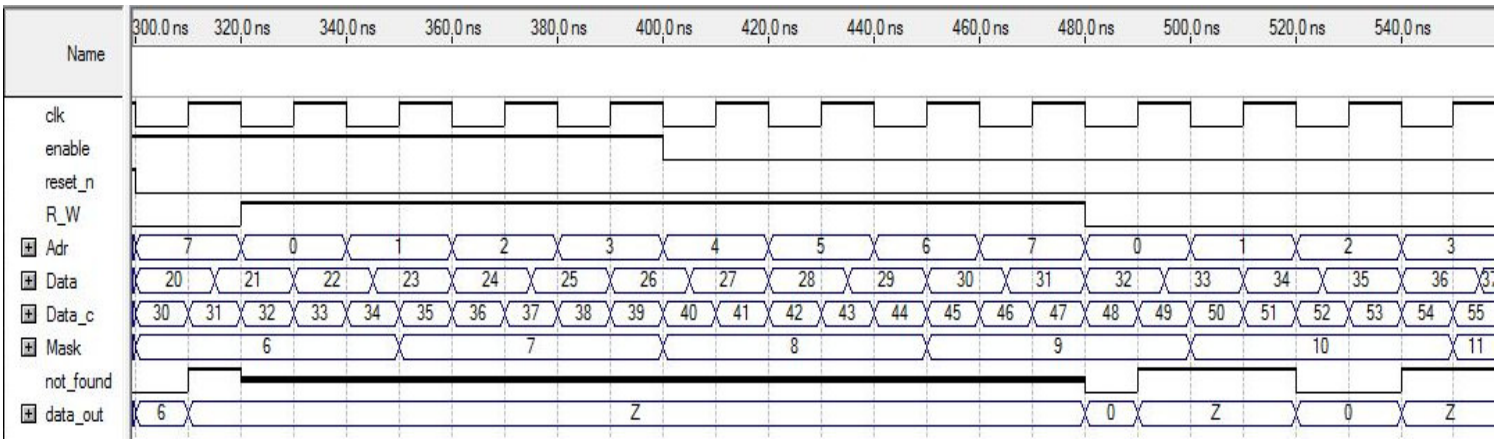

Figure 4.44: Schéma de simulation de la PIT « partie 2 »

La deuxième partie de la simulation où le **reset = '0'** qui définit que la **réinitialisation** est activée, l'entrée **Data** est alors mise à zéro et le **Data_c** est toujours comparée à **'0'** logique selon les bits spécifiés en **Mask**. nous avons remarqué que la sortie **Data-out** est à **'0'** si une correspondance est retrouvée, sinon c'est un état d'haute impédance. Comme montré dans la **Figure 4.44**.

Dans le cas de la mémoire PIT, seulement les 4 derniers bits qui sont pris en charge par la sortie **data_out** et cela dans le but de nous retourner à chaque fois le numéro de l'interface d'où il a reçu l'intérêt, afin de pouvoir renvoyer ce qu'il faut à cette interface spécifique au lieu de faire une diffusion pour le donnée vers toutes.

- D'après ces schémas de simulation la PIT est à présent validée car son fonctionnement est correct, elle sera traitée par le gestionnaire.

4.6. Implémentation du circuit sur la carte FPGA

Après avoir vérifié notre circuit, le compilé et synthétisé. Nous l'avons implémenté sur la carte FPGA. Dans ce qui suit nous expliquons les résultats obtenus et montrer la communication entre les deux cartes.

Nous présentons la structure des paquets que nous avons choisis pour ces tests, il y a six paquets dont trois entre eux sont des paquets **Interest**, et les trois restants sont leur paquets **Data** correspondants. Ainsi de cette façon les paquets correspondant portent le même nom dans leur champ **Name**.

Interest 1 : **B/systeme/embarque/ASIC/FPGA/VHDL**/Memoires/Rapides/un/seule/cycle

Data 1 : **A/systeme/embarque/ASIC/FPGA/VHDL**/Content Adressable Memory: TCAM

Interest 2 : **B/http/named/data/net/publication**/Technical/Presentations/meeting

Data 2 : **A/http/named/data/net/publication**/Information Centric Future: NDN

Interest 3 : **B/Youtube/Formation/Video/VHDL.RM**/FPGAs/Videos/and/VHDL/Solutions

Data 3 : **A/Youtube/Formation/Video/VHDL.RM**/Encoders using Logics Equations

4.6.1. Les résultats de l'exécution

Les résultats se divisent en deux grands cas qui sont : envoie d'un paquet **Interest** et envoie d'un paquet **Data**. Ensuite chacun possède deux autres cas particuliers.

4.6.1.1.Envoie d'un paquet Interest

Les deux cas particuliers de cette partie est selon l'existence de la donnée (paquet Data) correspondante dans le Content Store premiers cas est quand cette donnée n'existe pas et le deuxième est quand elle existe. Comme le montre la (**Figure 4.45**).

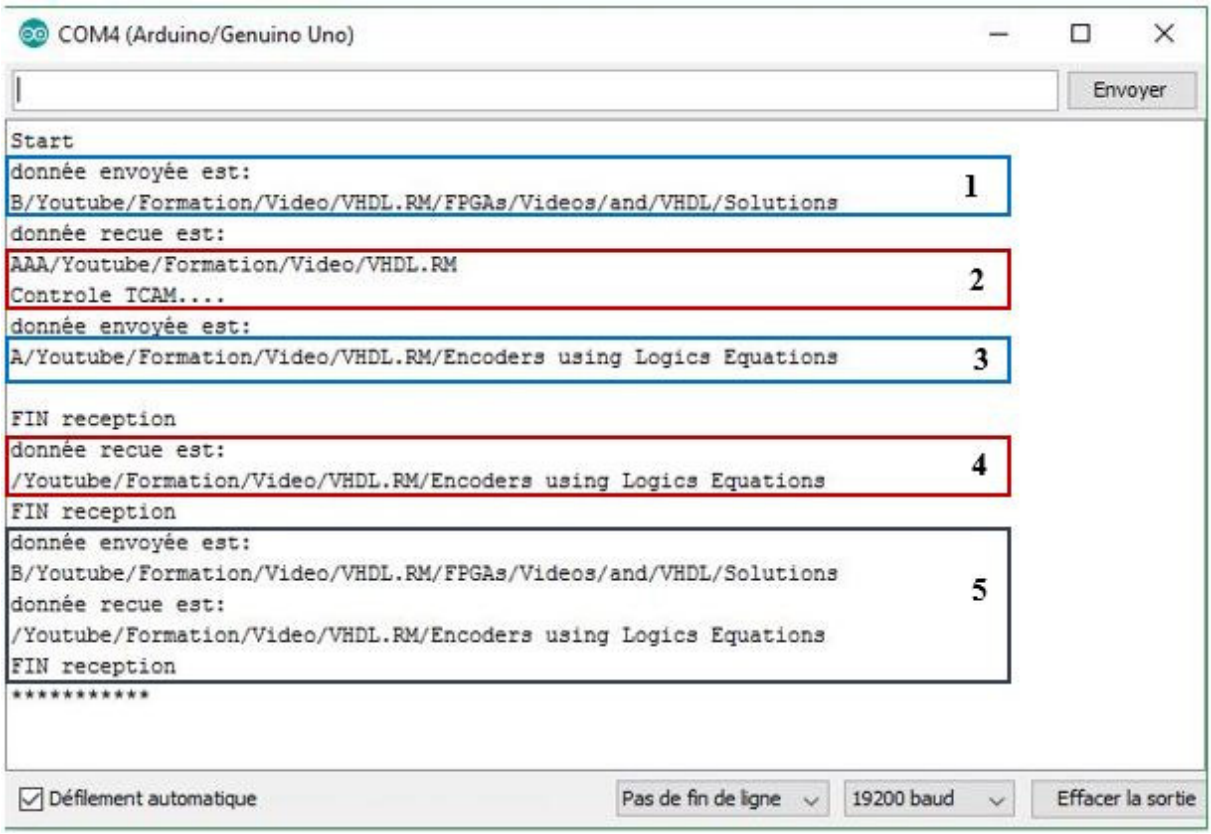

Figure 4.45: Résultats d'exécution - Exemple 1

Dans la **Figure 4.45**, nous distinguons cinq étapes que nous nous expliquons ci-dessous :

- ***Etape 1 :*** l'utilisateur **Arduino** désire une donnée particulière, il envoie un paquet Interest (**1**).
- ***Etape 2 :*** à la réception de l'intérêt par **FPGA**, une recherche dans la **CS** est effectuée sur le préfix de l'**Interest**. Dans ce cas le paquet Data n'existe pas dans la CS. Ensuite une autre recherche dans la **PIT** est effectuée sur le préfix de l'Interest et aucune entrée correspondante n'a été retrouvée, alors le paquet **Interest** reçu sera enregistré dans la

PIT. Une diffusion a été lancée (**2**) par le paquet : **AAA/Youtube/Formation/Video/VHDL.RM** pour pouvoir simuler Arduino comme étant un autre fournisseur de Data (comme une autre FPGA), et fait ainsi une ***controle_TCAM*** pour chercher si elle a la donnée pour répondre à cette diffusion.

- ***Etape 3 :*** dans cette étape on remarque que Arduino renvoie un paquet Data correspondant à l'Interest diffusé. Preuve qu'elle possède cette donnée et elle est transférée ensuite à l'FPGA afin de répondre à ces demandes stockées en PIT et l'enregistrer dans sa CS.
- ***Etape 4 :*** après la réception de la donnée de la part de FPGA, elle répond immédiatement aux demandes d'Interest existants dans sa PIT (**4**). D'ailleurs dans cette étape Arduino reçoit un paquet Data venant de FPGA et qui répond à l'Interest désiré en **étape 1**.

Ces quatre étapes résument le cas ou FPGA ne possède pas de la donnée dans sa CS, par contre la cinquième étape montre le deuxième cas.

- ***Etape 5 :*** **Arduino** envoie à l'FPGA une autre fois le même Interest de l'**étape 1**, et dans ce cas elle reçoit directement une réponse de sa part car la FPGA possède cette donnée dans sa **CS**.

4.6.1.2.Envoie d'un paquet Data

- **Cas 1 : Existence d'un paquet Interest correspondant dans la PIT**

Dans ce deuxième exemple nous supposons qu'il existe déjà un paquet Interest dans la PIT, et cela en l'envoyant d'abord en premier mais sans faire une diffusion. Ensuite un paquet data sera envoyé, comme le montre la (**Figure 4.46**)

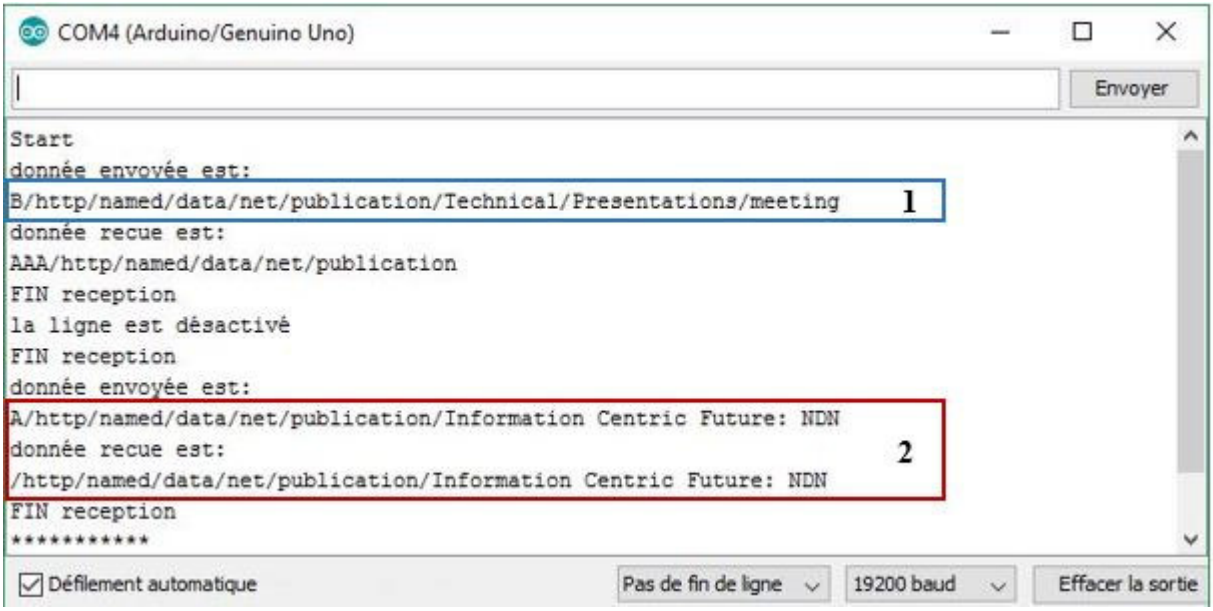

Figure 4.46: Résultats d'exécution - Exemple 2

- ***Etape 1 :*** Après l'envoi du paquet Interest (**1**), FPGA le met dans sa PIT jusqu'à ce qu'il soit servi par un paquet Data. Dans ce cas ce paquet est mis en PIT.
- ***Etape 2 :*** Arduino envoie ensuite un paquet Data vers FPGA et reçoit immédiatement une réponse pour l'Interest envoyé (**2**) une fois que l'FPGA a reçu la donnée (elle l'a met dans sa CS).

Cas 2 : Aucun paquet Interest correspondant dans la PIT

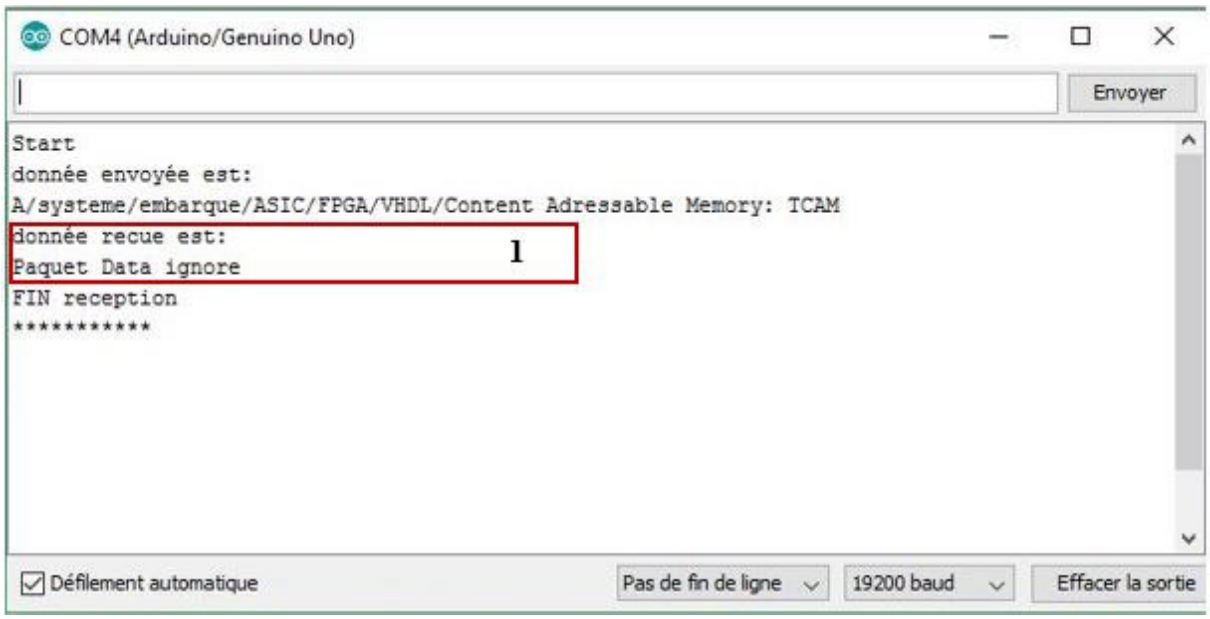

Figure 4.47: Résultats d'exécution - Exemple 3

Comme le montre la (**Figure 4.47**), lorsqu'un paquet Data est envoyé par **Arduino** et **FPGA** ne trouve aucune entrée correspondante dans la PIT, ce qui explique qu'aucune demande d'Interest n'a été effectuée pour cette donnée ou bien qu'il y avait déjà un Interest mais il a été supprimé car la demande a été traitée. Alors ce paquet Data reçu sera ignoré (**1**).

4.6.1.3. Le branchement réalisé pour programmer sur FPGA et Arduino

La **Figure 4.48** montre le branchement réalisé afin de programmer sur la carte FPGA :

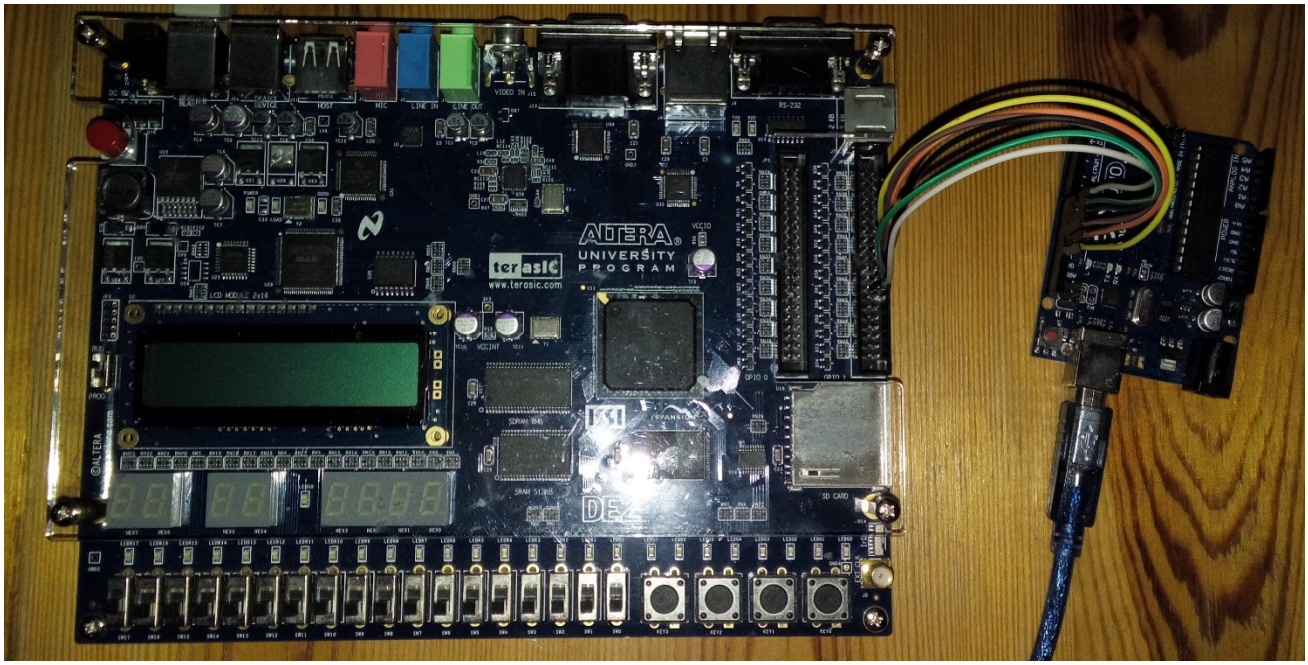

Figure 4.48: Branchement réalisé pour la carte FPGA et Arduino

4.7. Efficacité de la TCAM :

Afin de mettre le point sur l'utilité et l'efficacité de notre implémentation, nous avons fait un test qui permet de comparer entre une recherche d'une donnée dans une mémoire simple faite sur la carte Arduino, avec la recherche dans notre TCAM. Le temps de recherche est mesuré en milliseconde.

Cas 1 : Pour notre implémentation matérielle, la recherche se fait en un seul cycle d'horloge (un seul cycle de **CLK** de **FPGA**). Afin de calculer le temps que prend cette opération il suffit de définir la fréquence d'horloge que nous avons utilisée.

La fréquence de FPGA = 50 MHz. Avec la formule $T = \frac{1}{F}$,on calcule le temps comme suit :

$$T = \frac{1}{F} = \frac{1}{50} = \frac{1}{50\ 000\ 000} = 0.000\ 000\ 02\ (seconde) = 0.0002\ (milliseconde)$$

Cette valeur est valide pour n'importe quelle taille de la mémoire TCAM, car la recherche se fait en un seul cycle d'horloge. Cela est illustré par la **Figure 4.49.**

Cas 2 :

Sur l'Arduino, nous avons créé quatre tables de tailles différentes que nous avons parcourues pour rechercher une donnée selon le processus de recherche séquentiel classique. Les temps de recherches sont indiqués dans le **tableau 4.1**

La **figure 4.49** illustre la différence entre ces deux cas en terme de rapidité d'exécution

Tables	***Temps en milliseconde***
Table à huit lignes	***50***
Table à vingt lignes	***149***
Table à cent lignes	***845***
Table à deux cent lignes	***1647***

Tableau 4.2: Rapidité d'exècution du processus de recherche séquentiel

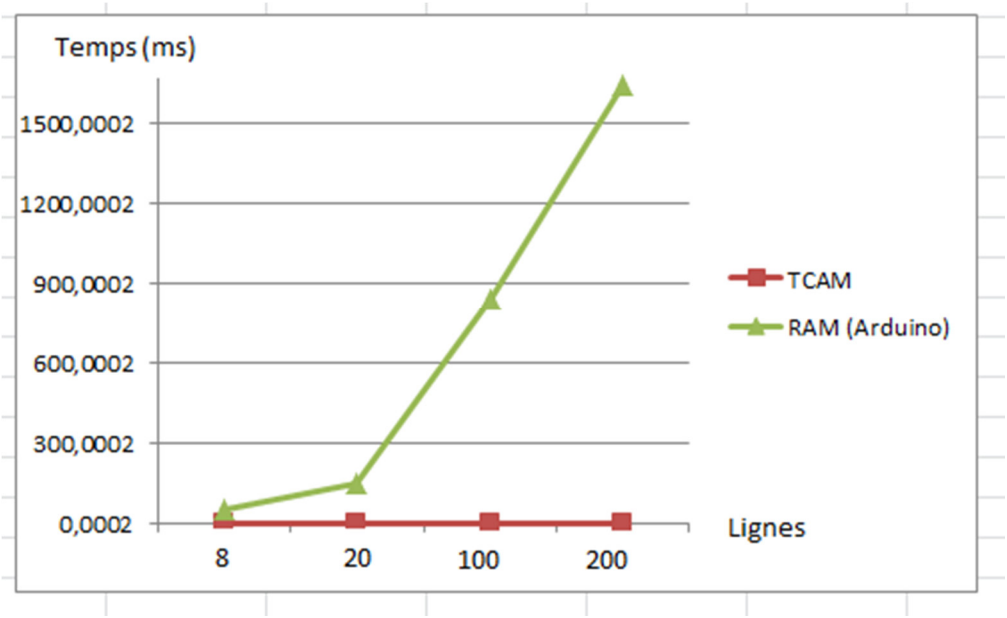

Figure 4.49: comparaison entre processus de recherche dans une TCAM et processus de recherche séquentiel

4.8. Conclusion

Dans ce chapitre, nous avons exposé les différentes étapes de conception et de développement de notre architecture à l'aide de la technologie **FPGA**, tout en présentant les outils utilisés. Ainsi pour la carte Arduino.

Pour cet effet, la suite été consacrée à l'étude d'une plateforme matérielle de gestion des mémoires TCAM et de la communisation SPI avec Arduino en utilisant la **DE2 Board**.

Le travail effectué nous a donné l'occasion d'une part de concevoir et de synthétiser une telle architecture. Et d'autre part, d'arriver à argumenter nos contributions avec les résultats approuvés, que nous jugeons satisfaisants. Qui montrent l'efficacité de notre implémentation en terme de la rapidité d'exécution.

Conclusion Générale

Conclusion générale

L'objectif de ce mémoire est la conception et la mise en œuvre d'une TCAM pour un routeur NDN, et au cours de ce travail nous avons fait une étude approfondie en première partie sur l'architecture NDN, afin de bien comprendre ses principes et son fonctionnement.

Une problématique a été traitée, à savoir : la gestion optimale des caches des routeurs NDN, qui consiste à la réduction de temps d'accès à ses mémoires. Pour cela nous avons fait des recherches sur les mémoires TCAM, pour déduire qu'elles sont les meilleures mémoires à utiliser dans ce type d'architecture réseau.

Après avoir créé les mémoires TCAM pour les caches à : **Content Store** et **Pending Interest Table**, et leurs simulations qui ont été satisfaisantes et qui répondent à nos besoins. Nous avons pu proposer une architecture et une gestion de ses deux caches selon un algorithme qui répond aux spécificités de NDN.

Enfin à la dernière étape de ce projet, nous avons réalisé une plateforme matérielle qui s'articule autour d'une carte **FPGA Cyclone II** d'**ALTERA DE2** et dans un environnement de développement **Quartus II**. Nous avons implémenté les mémoires TCAM (**CS** et **PIT**) sur le circuit de la carte FPGA, ainsi que leur gestionnaire qui permet de les faire fonctionner en suivant l'algorithme NDN. Nous avons connecté notre gestionnaire du routeur NDN avec un client Arduino à l'aide d'une communication série SPI

Ce travail nous a permis de maitriser le circuit FPGA, le langage VHDL. Ainsi que de bien comprendre la future architecture d'internet (NDN) et d'acquérir beaucoup de connaissances qui nous ont permis de s'approfondir dans le domaine de l'embarqués. Ces connaissances nous seront utiles pour d'autres projets au futur.

Les perspectives que l'on peut envisager pour faire suite à ce travail, est l'implémentation du troisième cache de NDN (**Forwarding Information Base**) qui définit la table de routage, et la gestion des caches peut être améliorée selon l'algorithme de routage choisit parmi ceux qui existent ou bien implémenter un nouveau protocole de routage pour NDN

Bibliographie

[1] Haowei Yuan **« Data Structures and Algorithms for Scalable NDN Forwarding »** Washington University in Saint Louis, Décembre 2015.

[2] http://named-data.net/

[3] Faizul Bari, Shihabur Rahman Chowdhury, and Reaz Ahmed, University of Waterloo Raouf Boutaba, University of Waterloo and Pohang University of Science and Technology Bertrand Mathieu, Orange Labs **« A Survey of Naming and Routing in Information-Centric Networks »**, Décembre 2012.

[4] Elian Aubry **«Protocole de routage pour l'architecture NDN»** Thèse de doctorat de l'université de Lorraine, 19 décembre 2017.

[5] https://community.cisco.com/t5/network-architecture-documents/cam-content-addressable-memory-vs-tcam-ternary-content/ta-p/3107938

[6] https://www.pagiamtzis.com/cam/camintro/

[7] Layla Hattim Abood, **«FPGA Implementation of Ternary Content Addressable Memory »**

[8] https://etherealmind.com/basics-what-is-ternary-content-address-memory-tcam/

[9] Ergun comert **«SPI Interface with VHDL »** OKAN University, 2011

[10] Mickaël Dardaillon **« Introduction aux FPGA »**, 2011

[11] Nicolas MARQUES **« Méthodologie et architecture adaptative pour le placement efficace de taches matérielles de tailles variables sur des partitions reconfigurables »** Thèse de doctorat de l'université de Lorraine, 26 novembre 2012.

[12] Pong P. Chu, «EMBEDDED SOPC DESIGN WITH NIOS II PROCESSOR AND VHDL EXAMPLES » Cleveland State University

[13] « DE2 Development and Education Board User Manual », Altera Corporation, 2006 Version 1.4

[14] Pong P. Chu «FPGA PROTOTYPING BY VHDL EXAMPLES» Cleveland State University.

[15] LECHALUPÉ Julien **« Cours d'initiation à Arduino »,** Université Paul Sabatier, Mai 2014.

Liste des figures

Résumé

Résumé

L'objectif de ce travail est d'évaluer la possibilité de diminuer le temps d'accès aux données dans le réseau NDN (Named data networking). La nouvelle architecture d'internet NDN souffre de surcharge de donnée au niveau de son cache et cela rend le processus de recherche très long. L'objectif du cache est d'enregistrer les données au niveau du routeur pour les garder plus proche de l'utilisateur. En revanche, le grand volume d'échange sur l'internet pose le problème de mise à l'échelle car le temps de recherche dans le cache devient important et, par conséquent, son objectif n'est pas atteint. Notre travail explore la possibilité de diminuer le temps de recherche de donnée au niveau de cache de routeur **NDN** en utilisant un cache matériel conçu par les technologies de mémoire **TCAMs**.

Pour répondre à la problématique, une architecture complète de routeur **NDN** a été réalisée. Les caches des routeurs sont remplacés par des mémoires spéciale appelé TCAM et ensuite géré par un algorithme approprié. Les résultats obtenus montrent bien que le temps d'accès aux données est très court par rapport aux mémoires standards.

Au vu des résultats de tests, nous constatons que, l'architecture et l'algorithme proposé peut répondre aux objectifs du caching NDN. Le réseau NDN déployé avec notre solution de caching matériel aura un temps de réponse bien meilleur que les mémoires standards.