MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE



**THESE**

Présentée pour l'obtention du diplôme de MAGISTER

**SPECIALITE**

Signaux et Systèmes

**OPTION**

Systèmes de Traitement de l'Information

**THEME**

# CONCEPTION D'UN CRYPTOPROCESSEUR A BASE DE L'ALGORITHME D.E.S

**PAR**

**ADROUCHE Djamel**
Ingénieur d'Etat, EMP (ex: ENITA)

Soutenu le : 03 / 07 / 2001

Devant le Jury :

| | | | |
|---|---|---|---|
| **Président** : | D.BERKANI | Professeur | ENP |
| **Rapporteurs** : | A. FARAH | Professeur | ENP |
| | R. SADOUN | Chargé de cours | ENP |
| **Examinateurs** : | A. BELOUCHRANI | Docteur | ENP |
| | L. HAMAMI | Chargé de cours | ENP |
| | H. BOUSBIA | Chargé de cours | ENP |

Ecole Nationale Polytechnique 10, Avenue Hassen Badi, El-Harrach, Alger

# Remerciements



Je tiens à remercier vivement :

- Dieu le tout puissant pour tout ce qu'il ma donné.
- Mon père, ma mère, ma femme, mes frères et sœurs pour leurs encouragements et leur patience.
- Mes directeurs de thèse, le professeur FARAH de m'avoir proposé ce sujet, Monsieur SADOUN d'avoir dirigé ce travail et pour tous ses précieux conseils.
- Monsieur BERKANI de l'honneur qu'il me fait en acceptant de présider le jury de cette thèse.
- Madame HAMAMI, Messieurs BELOUCHRANI et BOUSBIA de l'honneur qu'ils me font en acceptant de faire partie de ce jury.
- Les enseignants, l'administration et le personnel de l'Ecole Nationale Polytechnique.

# Sommaire

# Listes des Figures

A.4 Modèle physique de l'implémentation de l'algorithme DES, avec une structure en pipeline à deux étages (DES_VER3.1), sur le circuit FPGA XC4013XL-3BG256

A.5 Modèle physique de l'implémentation de l'algorithme DES, avec une structure combinée (DES_VER4), sur le circuit FPGA XC4013XL-3BG256

A.6 Modèle physique de l'implémentation du cryptoprocesseur sur le circuit FPGA XC4013XL-3BG256

B.1 Conception DES_VER1.1 (structure de base)

B.2 Conception DES_VER2.1 (deux expansions de la boucle)

B.3 Conception DES_VER2.2 (quatre expansions de la boucle)

B.4 Conception DES_VER3.1 (deux pipelines)

B.5 Conception DES_VER3.2 (quatre pipelines)

B.6 Conception DES_VER4 (structure combinée)

C.1 Fonctionnement du cryptoprocesseur en mode d'adressage interruptible

D.1 Fonctionnement du cryptoprocesseur en mode d'adressage indexé

D.2 Fonctionnement du cryptoprocesseur en mode d'adressage immédiat

D.3 Fonctionnement du cryptoprocesseur en mode d'adressage direct

D.4 Fonctionnement du cryptoprocesseur en mode d'adressage indirect

E Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur

F Caractéristiques du cryptoprocesseur obtenu

# Listes des tableaux

# Introduction



Le monde d'aujourd'hui évolue inexorablement vers un modèle de société dominé essentiellement par l'information.
L'apparition des autoroutes de l'information et leur utilisation dans le grand public, de même que l'utilisation quotidienne des procédés de traitement automatique de l'information provoquent de plus en plus de problème de sécurité [1].

Des expériences de porte-monnaie électronique sont en cours, le réseau Internet explose, l'usage du courrier électronique pour communiquer s'étend toujours plus, les cartes à puces deviennent une réalité. D'autres domaines des communications sont aussi en pleine croissance. On cite pour exemple la communication sans fil.
En un mot, les moyens de payement et de communication évoluent à grande vitesse.

Dans toutes ces applications, la vitesse, la confidentialité, la sécurité et l'intégrité sont des composantes majeures. Sans celles ci, ces applications sont vouées à l'échec ou aux abus les plus déplorables.

Capter une conversation sur téléphone mobile, pirater le code d'une carte de crédit, falsifier les transactions bancaires ne sont que peu d'exemples des menaces que peut subir une infrastructure de communication non protégée. Les principaux outils mis en œuvre pour assurer la sécurité désirée découlent de la cryptographie.

On peut classer les méthodes qui s'y rattachent en deux grandes classes (on passera sous silence la troisième classe, appartenant à celles des méthodes du futur, dite cryptographie quantique. Elle est née au début des années 70 et repose sur le principe d'incertitude d'Heisenberg. Les systèmes qui en découlent sont toujours à un stade expérimental [2]).

La première classe, dite cryptographie classique, décrit la période avant les ordinateurs. Elle traite des systèmes reposant sur les lettres et les caractères d'une langue naturelle. Les principaux outils utilisés remplacent des caractères par des autres et les transposent dans des ordres différents. Les meilleurs systèmes (de cette classe d'algorithmes) répètent ces deux opérations de base plusieurs fois. Cela suppose que les procédures (de chiffrement ou déchiffrement) soient gardées secrètes et sans cela, le système est complètement inefficace (n'importe qui peut déchiffrer le message codé). On appelle généralement cette classe de méthodes : le chiffrement à usage restreint.
Ces méthodes n'ont plus qu'un intérêt historique, car dans la pratique, elles sont très vite cassées par les cryptanalystes.

Les méthodes utilisées de nos jours, appartenant à la classe de la cryptographie dite moderne, sont plus complexes. Elles ne peuvent être mises en œuvre directement par les humains. Des algorithmes complexes ont été conçu et sont exécutés sur des ordinateurs.
Cependant, la philosophie reste la même. La différence fondamentale est que les méthodes modernes (les algorithmes, puisque l'on utilise maintenant des ordinateurs) manipulent directement des bits (liés à l'implantation sur les machines) contrairement aux anciennes méthodes qui opéraient sur des caractères alphabétiques. Ce n'est donc qu'un changement de taille (ou de représentation), puisque l'on utilise plus que deux éléments au lieu des lettres de l'alphabet cible. La plupart des bons systèmes de cette catégorie combinent toujours des substitutions et des transpositions, et les règles sont connues de tous ; c'est pourquoi on appelle cette classe, le chiffrement à usage général.
La sécurité de ces méthodes reposent maintenant sur un nouveau concept clé : les clés.
La manière de générer et de gérer ces clefs a conduit au développement de deux types de système : les systèmes symétriques et les systèmes dits asymétriques.

Les systèmes symétriques sont synonymes de systèmes à clés secrètes. Une même clé est utilisé pour le chiffrement et le déchiffrement, d'où l'obligation que celle-ci reste confidentielle, sous peine de rendre le système inéfficient. Dans cette catégorie, le D.E.S. (Data Encryption Standard) est un standard mondial en cours depuis plus de 15 ans . Il a été développé en 1976 par IBM pour le N.B.S. (National Bureau of Standards) avec pour objectif de fournir un nouveau standard de fait destiné à limiter la prolifération d'algorithmes différents qui ne pouvaient pas communiquer entre eux. Bien qu'il montre des signes de vieillesse, il a remarquablement bien résisté à des années de cryptanalyse et il est toujours sûr contre tous les adversaires excepté peut-être les plus puissants. Il est devenu le système de chiffrement le plus utilisé dans le monde. Depuis son adoption, le D.E.S. a été réévalué environ tous les cinq ans.
Il reste très utilisé dans le domaine commercial et des banques, et il est implanté dans de nombreuses cartes de crédits dédiés (smart cards, système électronique de communication). Son principal avantage est d'offrir une vitesse de chiffrement et déchiffrement élevée.

Les systèmes asymétriques sont synonymes d'algorithmes à clés publiques. Le concept a été inventé en 1975 par deux ingénieurs en électronique : Whitfield Diffe et Martin Hellman de l'Université de Stanford. Une clé différente est utilisée à la fois pour chiffrer et déchiffrer, et il est impossible de générer une clé à partir de l'autre. Il résout parfaitement le problème majeur d'échange de clefs qui est posé par les algorithmes à clefs secrètes.
La première application de ce principe fut le chiffrement RSA. Cet algorithme a été inventé par Rivest R., Shamir A., et Adlemann L. du M.I.T. (Massachussets Institute of Technology). C'est l'algorithme à clé publique le plus commode qui existe. Comme pour le D.E.S. sa sécurité repose sur l'utilisation de clés suffisamment longue (512 bits n'est pas assez, 768 est modérément sûr, et 1024 bits est une bonne clé). C'est la difficulté que l'on a à factoriser les entiers premiers qui font que l'on ne peut que difficilement casser cet algorithme. Cependant de larges avancées en matière de

factorisation des entiers larges [1], ou une augmentation considérable de la puissance des super-calculateurs rendront RSA très vulnérable.
RSA est aujourd'hui utilisé dans une large variété de produits (téléphones, réseaux Ethernet, etc...), de logiciels de différentes marques (Microsoft, Apple, Novell, Sun), dans des industries et enfin dans les télécommunications.

Les algorithmes à clé publique sont assez lents. La méthode généralement utilisée pour envoyer un message, est de tirer au hasard une clé secrète, chiffrer le message avec un algorithme à clé privée en utilisant cette clé, puis chiffrer cette clé aléatoire elle-même avec la clé publique du destinataire. On aborde ainsi le cas des systèmes de chiffrement mixte. Ceci permet d'avoir la sécurité des systèmes à clé publique, avec la performance des systèmes à clé privée. Le célèbre PGP de Phil Zimmermann constitue un excellent exemple. Correctement utilisé, il est sûr, même contre les meilleurs cryptanalystes du monde (c'est d'ailleurs pour cette raison que son utilisation est formellement interdite dans divers pays dits développés).

Pour des raisons de performance et de sécurité, il est souvent nécessaire d'implémenter les algorithmes cryptographiques sur des dispositifs matériels spécialisés réalisé sur des puces résistant à l'investigation dont l'exemple le plus célèbre reste la puce Clipper.

Les solutions ASIC traditionnelles possèdent l'inconvénient de réduire la flexibilité comparés aux solutions logicielle. Une autre alternative pour l'implémentation des algorithmes cryptographiques est d'isoler le stockage et l'exécution dans un périphérique matériel tel que les cryptoprocesseurs, les cartes à puces, les cartes PCMCIA et les boites "noires" résistantes à tout les types d'attaques.
D'un autre côté, les protocoles de sécurité modernes sont de plus en plus définis pour être à algorithme indépendant. Un degré élevé de flexibilité est exigé pour ces standards et applications cryptographiques. Une solution prometteuse qui combine un degré élevé de flexibilité avec une vitesse et une sécurité physique des composants électroniques traditionnelle est l'implémentation des algorithmes cryptographiques sur des composants reconfigurables tel que les circuits FPGAs (Field Programmable Gate Arrays). Les composants FPGAs peuvent interchanger les algorithmes de chiffrement. Ceci veut dire que le même composant peut être utilisé pour différents types d'algorithmes, bien qu'à un moment donné un seul algorithme est configuré.
En résumé, l'implémentation des algorithmes cryptographiques sur les circuits FPGAs nous offres un certains nombres d'avantages tels que :

- Le même composant FPGA peut être reconfiguré à la volée afin de supporter différents types d'algorithmes, par exemple, les algorithmes à clefs publiques et les algorithmes à clefs privés,
- Le même composant FPGA peut convenir pour différentes versions d'un même algorithme (exemple, DES et triple-DES),
- Les paramètres architecturaux du composant FPGA sont altérables, donc des fonctions de caractéristiques différentes peuvent être facilement réalisées.

Dans le cadre qui nous a été défini, nous nous intéresserons à l'implémentation d'algorithme symétrique (le DES) sur une structure microprocesseur à base de FPGA.

Les travaux antérieurs concernants les implémentations matérielles d'une manière générale et de l'algorithme DES d'une façon particulière ont fait l'objet d'une distinction entre les implémentations VLSI et FPGA. Ils indiquent quelles sont les technologies futures qui peuvent paraître très intéressantes pour l'implémentation des circuits cryptographiques.

Dans la conférence *CRYPTO 84*, deux thèmes ont abordés le domaine de l'implémentation des algorithmes cryptographiques sur différentes technologies de circuits. La première communication de J. Goubert, F. Hoornaert et Y. Desmedt [3] décrit l'implémentation de l'algorithme DES sur un circuit sur mesure "FULL CUSTOM". Ce circuit ASIC supporte les quatre modes de fonctionnements du DES (cf. paragraphe 4.4). La vitesse maximum atteinte par ce circuit est de **20 Mbit/s**. Le deuxième article de A. Matusevich, R.C. Fairfield et J. Plany [4] concerne l'implémentation d'un cryptoprocesseur sur un circuit de technologie LSI. Il offre à l'utilisateur la possibilité de programmer n'importe quel mode de fonctionnement de l'algorithme DES. La vitesse maximale de fonctionnement de ce circuit est de **4.72 Mbit/s**.

En 1988, J. Vandewalle, I. Verbauwhede, F. Hoornaert, et H.J. De Man [5] avaient publié un article décrivant l'implémentation de l'algorithme DES sur un composant **CMOS à 3-µm** avec une technologie double métal. Le circuit peut atteindre un débit de **32 Mbit/sec**. Il possède une vitesse de 60% plus élevée par rapport aux travaux de l'article [3] et supporte aussi tout les modes de fonctionnement de l'algorithme DES.

Toujours dans le domaine de l'implémentation des algorithmes cryptographiques, une ancienne référence de A.G. Broscius et J.M. Smith [6] considérée comme la première étude qui s'est intéressé essentiellement à l'accroissement des performances de l'algorithme. Le principe consiste à structurer l'architecture de l'algorithme DES afin d'en tirer d'autres options architecturales qui permettent d'accélérer le traitement des données. Cet article cite aussi d'autres techniques, par exemple, l'utilisation d'une itération pour calculer à l'avance une sous-clef. Le débit de cette implémentation n'a pas été mentionné.

Les implémentations matérielles actuelles de l'algorithme DES peuvent atteindre un débit de **1Gbit/sec** et au-delà.

Les travaux de recherche de H. Eberle et C.P. Thacker [7] est le premier rapport indiquant l'utilisation d'un composant à base d'une technologie moderne de Gallium d'Arsenide pour atteindre un débit de 1 Gbit/s. Plus tard dans une autre publication, du même groupe de recherche [8], décrit cette conception plus en détail. Ils ont aussi mentionnés que le composant le plus rapide testé peut fonctionner jusqu'à un débit de

**1.4Gbit/sec**. L'inconvénient majeur de cette conception, est le fait qu'un port de 7 bits de largeur est disponible pour le chargement de la clef principale.
Ceci veut dire que l'échange fréquent de clefs fait chuter considérablement la vitesse de ce composant.

La publication [9] de Jens Kaps et Christof Paar de l'Institut Polytechnique de Worcester des Etats Unies d'Amérique décrit l'implémentation de l'algorithme DES sur le composant FPGA **XC4028EX-3-PG299**. Cette implémentation utilise une architecture en pipeline à quatre (04) étages. Elle réalise des opérations de chiffrement/déchiffrement avec un débit de **384 Mbit/sec**.

Un article publié par Xilinx, Inc, [10] traite de l'implémentation de l'algorithme DES sur l'architecture VIRTEX, famille des composants FPGA. Cette dernière est considérée comme successeur de la famille XC4000. Les caractéristiques de la structure interne des CLBs [11] du composant Virtex, le rendent particulièrement adapté à l'algorithme DES :

- Chacune des huit tables-S est représentée par une table à 6 bits en entrée et 4 bits en sortie. Des algorithmes de minimisation des structures logiques ont réussi à trouver des optimisations dans la structure des tables-S. Il est plus raisonnable d'implémenter chaque table-S sous forme de 4 générateurs de fonctions (LUTs) dont chacune possède 6 bits en entrée et 1 bit en sortie. Un seul CLB de l'architecture Virtex suffit pour implémenter un générateur de fonction (LUT) à 6 entrées. Ceci est rendu possible par la combinaison de quatre LUTs à 4 entrées avec des multiplexeurs internes au CLB. L'implémentation d'un LUT à 6 entrées sur un XC4000 est plus lente, puisqu'elle nécessite deux CLB.
- Dans la famille Virtex, un LUT à 4 entrées peut fournir la fonction d'un registre à décalage variant de 1 à 16 bits (SRL16). Les entrées du LUT définissent sur le registre SRL16 la position de la sortie désirée. Au delà, de 3bits la puissance consommée par le SRL16 est inférieure à celle consommée par une série de bascules connectées entre elles. Le SRL16 fournit un haut degré de pipeline. Ce dernier peut être utilisé dans le chemin de données de l'algorithme DES.

L'implémentation sur un composant **Virtex V400-6** a donné un débit de **3.6 Gbit/sec**.

Le standard IPsec, protocole des transmissions chiffrées, est utilisé dans la plupart des réseaux virtuels privés. Ce protocole exige que les paquets Internet doivent être chiffrés avec le triple DES en mode Cypher Block Chaining (CBC). Mais l'implémentation du **3DES-CBC** sur des circuits statiques **CMOS** avec une technologie 0.25u [12] empêche de chiffrer un paquet de donnée avec un débit supérieur à **750 Mb/s**. Pour pouvoir utiliser ces circuits dans les réseaux Ethernet gigabit, plusieurs paquets doivent être chiffrés en parallèle. Ce circuit traite deux itérations de l'algorithme DES pendant chaque cycle d'horloge. Chaque cycle dure 1.25ns, et 24 cycles d'horloges sont nécessaires pour achever les 48 itérations du triple DES, donc un bloc de données de 64 bits est chiffré à travers le 3DES CBC durant chaque 30ns. Avec cette technique un débit de **2.1Gb/s** est facilement atteint.

La thèse de J.Põldre [13] du département d'informatique, de l'université technique de Tallinn. Avait pour objectif l'implémentation d'un cryptoprocesseur sur un composant FPGA réalisant les fonctions cryptographiques suivantes : chiffrement de données en utilisant l'algorithme DES et échange de clefs grâce à l'algorithme à clef publique de Rivest, Shamir et Adelman (RSA). Ce cryptoprocesseur fonctionne avec une vitesse maximale de **25MHz**.

Un autre article datant de janvier 2000, publié par CAST Inc, [14] annoncait la conception d'un cryptoprocesseur X_DES implémenté sur un composant **Virtex V150-6**. Ce cryptoprocesseur fonctionnant à une fréquence d'horloge de **101MHz** est adapté aux quatre modes de fonctionnement du DES et convient particulièrement au triple DES.

Des implémentations type smart card sont aussi en cours. Elles visent les segments non négligeables des systèmes transactionnels. On cite l'article « Implementation of Four AES Candidates on Two Smart » de Cards Gael, Hachez François Koeune et Jean-Jacques Quisquater de l'Université catholique de Louvain - 1998 - qui visait la comparaison de diverses implémentation cryptographiques sur des plates formes 8 bits.

En ce basant sur les considérations générales des paragraphes précédant, nous avons définis les buts à atteindre à travers notre travail de thèse:

1. Arriver à implémenter sur un composant FPGA un cryptoprocesseur réalisant les fonctions de base d'un microprocesseur en plus d'une fonction spécifique dédiée au chiffrement et au déchiffrement des données.
2. Etudier et choisir un algorithme cryptographique pour la réalisation de la fonction de chiffrement/déchiffrement du cryptoprocesseur.
3. Implémenter l'algorithme correspondant sur un composant FPGA tenant compte des diverses options architecturales. Toutes ces options architecturales découlent de la structure de base de l'algorithme choisi. Elles visent à augmenter les performances absolues souhaitées.
4. Donner la possibilité d'interchanger plusieurs algorithmes cryptographiques pour la réalisation de la fonction de chiffrement/déchiffrement du cryptoprocesseur.
5. Donner la possibilité d'étendre les fonctionnalités du cryptoprocesseur par l'ajout d'autres fonctions cryptographiques, à savoir, échange de clefs, authentification et signature numérique.

## Plan de la thèse

Notre thèse a été décomposée en six chapitres.

**L'Introduction** donne une brève introduction de cette thèse et décrit les motivations pour la réalisation des travaux associés et justifie aussi les raisons des décisions prises dans certaines directions. Elle contient une rétrospective des travaux antérieurs entrepris dans le domaine de l'implémentation des cryptoprocesseurs sur les composants FPGAs et d'une manière générale l'implémentation des algorithmes cryptographiques.

**Le chapitre I** fait l'objet d'une étude théorique et ce en abordant les aspects suivants :

- La méthodologie de conception et le cycle d'implémentation. En plus, d'une vue générale des outils matériels et logiciels utilisés lors de nos travaux, cette partie inclut aussi certaines remarques sur les performances et l'efficacité des outils de conception utilisés.

- L'étude théorique de la cryptographie et de ses concepts de bases. Ce chapitre justifie aussi certains choix adoptés, par exemple, le choix de l'implémentation matérielle au lieu d'une implémentation logicielle et l'adoption des algorithmes à clef secrète pour le chiffrement des données.

- L'introduction au Standard de Chiffrement de Données ainsi que les modes de fonctionnement de l'algorithme DES et les améliorations qui lui ont été apportées.

**Le chapitre II** est consacré à l'implémentation de l'algorithme DES sur FPGA. Ce chapitre traite tout le cycle de conception, à savoir, le modèle comportemental, le modèle structurel et enfin le modèle physique. A cet effet, ce chapitre est divisé en trois grands sous-chapitres :

- **Modèles comportementaux de l'algorithme DES** : commence par explorer les différentes options architecturales du DES, tel que l'architecture en boule expansée et en pipeline. Ensuite, il donne un résumé des différentes versions architecturales que nous avons décidé d'implémenter.

- **Modèle structurel associé à l'algorithme DES** : il est consacré à la conception du circuit. Afin de réaliser des optimisations sur les différentes conceptions, l'algorithme DES est divisé en plusieurs blocs fonctionnels

- **Implémentation du modèle structurel associé à l'algorithme DES** : traite en détail des architectures implémentées et donne une vue globale des codes sources. Les signaux de la logique de contrôle de chaque architecture sont expliqués en détail.

**Le chapitre III** est consacré à l'implémentation du cryptoprocesseur sur FPGA. Il est scindé en deux sous-chapitres. Ces derniers sont consacrés respectivement au modèle comportemental et structurel du cryptoprocesseur.

**Modèle comportemental du cryptoprocesseur** : donne le modèle comportemental du circuit à concevoir et les caractéristiques envisagées.

**Modèle structurel du cryptoprocesseur** : détaille les deux (02) parties formants le cryptoprocesseur, à savoir : le chemin de données et le séquenceur.

**Le chapitre IV** présente les résultats obtenus par l'implémentation de l'algorithme DES, de ses divers options architecturales ainsi que ceux du cryptoprocesseur. Il fait aussi l'objet d'une comparaison entre les performances obtenues par l'implémentation des divers architectures , à savoir, les architectures en boucles expansées et en pipeline.

**Enfin, une conclusion** qui résume les résultats obtenus et propose quelques recommandations pour les travaux futurs.

# Chapitre I

# Etude théoriques

## I.1 Introduction

En vue d'atteindre l'objectif qui nous a été assigné ; à savoir ; concevoir un processeur dédié et réalisant les fonctions essentielles d'un microprocesseur complétées par des fonctions spécialisées dans le domaine de la cryptographie. Ces fonctions s'expriment par une unité spécialisée dans le chiffrement et le déchiffrement et sera réalisée à base d'un l'algorithme cryptographique préalablement choisi. A cet effet, nous aborderons dans ce chapitre les notions théoriques suivantes :

1. L'implémentation du cryptoprocesseur et en particulier son bloc de chiffrement/déchiffrement sur un composant FPGA nous impose la maîtrise des techniques de synthèse architecturale, la méthodologie de conception pour la technologie cible ainsi que les outils de synthèse correspondants.

2. Etant donné que le cœur du module de chiffrement et de déchiffrement du cryptoprocesseur est à la base d'un algorithme cryptographique ceci nous amène à connaître les fondements de la cryptographie, notamment les protocoles, les techniques et les algorithmes cryptographiques.

3. Le choix du type d'implémentation (matérielle ou logicielle) et de l'algorithme assurant la fonction de chiffrement et de déchiffrement, étant justifié par l'étude portant sur les principes et les techniques de la cryptographie, nous consacrerons la suite du chapitre à l'étude détaillée de l'algorithme DES pour son implémentation dans une structure microprocesseur.

# I.2 - Méthodologie de conception

## I.2.1 Introduction

Les techniques et les capacités d'intégration de circuits électroniques ont beaucoup progressé ces dernières années. Cette évolution permet la réalisation de circuits et de systèmes matériels toujours plus complexes.

La méthode de conception traditionnelle de composants consistait tout d'abord à transformer un cahier des charges définissant les fonctionnalités du système en spécifications. Ces spécifications sont alors utilisées pour générer l'architecture du système en terme d'interconnexions de blocs fonctionnels. Chaque bloc est alors défini au niveau portes logiques. Le schéma ainsi obtenu sert de document pour la maintenance du système.

La conception de circuits de grande complexité suppose de partir de spécifications abstraites. Il est en effet plus aisé pour un concepteur, de manipuler des fonctions que de dessiner transistor par transistor le schéma de son circuit. Idéalement, on souhaite fournir aux outils de conception des algorithmes exprimés de manière fonctionnelle, et en obtenir le plan de masse des circuits. L'utilisation de tels outils de haut niveau permet en outre, de fiabiliser le processus de conception. Ils génèrent également un gain de temps significatif dans le processus de conception. Le temps ainsi économisé peut être utilisé pour réduire le délai de mise sur le marché du produit ou pour augmenter la qualité des circuits par l'exploration d'autres solutions matérielles.

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

La modélisation (qui est l'action de décrire le comportement d'un système dans son ensemble) modifie en profondeur les méthodes de conception.

Parmi les langages de description du matériel disponibles, VHDL est celui qui supporte le plus haut niveau d'abstraction dans la description d'un système, tout en acceptant les descriptions de bas niveau.

La modélisation de circuits aussi complexes qu'un processeur est un véritable défi pour les concepteurs de modèles VHDL. Dans certains cas, l'étape de modélisation peut être suivie de la synthèse des spécifications fonctionnelles en vue de reproduire le circuit original en utilisant de nouveaux outils de synthèse associés à des technologie plus récente.

## I.2.2 Expression d'une architecture

Un circuit est destiné à résoudre un problème. On peut donc voir ce circuit d'une manière plus ou moins proche de la réalité physique ou de l'aspect algorithmique.
On distingue trois angles de vue (plus couramment désignés par les termes domaines d'expression pour exprimer une architecture (figure I.2.1).

Ces domaines d'expression sont eux-mêmes découpés en cinq niveaux de description plus ou moins abstraits par rapport aux détails du circuit. Les cinq degrés d'abstraction du domaine comportemental par exemple sont : interrupteur, logique, transfert de registres, algorithme et système. Les niveaux système et algorithmique sont quelquefois regroupés dans un seul niveau .

Les trois domaines d'expression sont [15] :

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

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

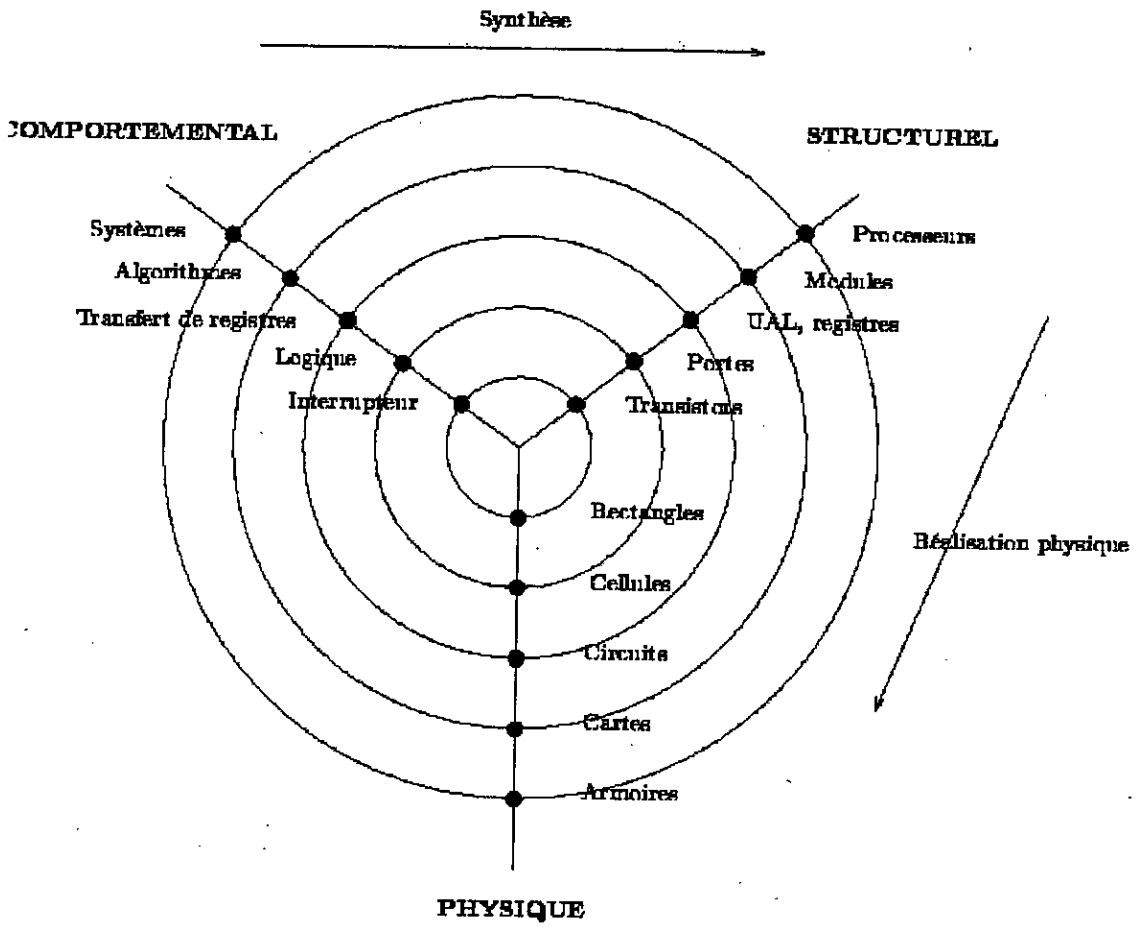



Figure I.2.1 : Diagramme en Y introduit par Gajski décrivant les trois vues d'une architecture.

## I.2.3 Spécification de systèmes

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

- Large choix de technologies. Le concepteur de systèmes doit faire face à une palette de choix de conception en constante augmentation (technologies de fabrication, méthode de conception, choix architecturaux).

- Partage matériel/logiciel. Les systèmes comportent une part de plus en plus significative de logiciel, en raison de la disponibilité de cœurs de processeurs (RISC, DSP, micro-contrôleurs), comme éléments de bibliothèque. Il s'agit de pouvoir réaliser une spécification indifféremment sous forme matérielle ou logicielle, de pouvoir déterminer le partitionnement et d'engendrer automatiquement des interfaces.

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

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

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

### I.2.3.1 Niveau interrupteur (circuit)

Une description au niveau interrupteur manipule des composants de très bas niveau caractérisés par des délais. Ces différents éléments sont regroupés pour former le circuit. Il n'est pas raisonnable de spécifier un système complexe à ce niveau de nos jours.

### I.2.3.2 Niveau logique

Une spécification logique est un ensemble de fonctions booléennes ou de machines d'états [16]. Là encore son utilisation se limite à de petits circuits.

### I.2.3.3 Niveau transfert de registre

Le niveau supérieur est appelé transfert de registres ou RTL (*Register Transfer Level*). La description porte sur les états des registres et les transferts d'états. On distingue trois grandes parties : la partie chemin de données qui représente les calculs, la partie contrôle, et la partie mémoire. Plusieurs langages permettent de décrire un circuit au niveau RTL.

VHDL [17, 18, 19, 20, 21, 22] a été introduit par le département de la défense américain. Il répond au besoin de standardisation exprimé il y a une dizaine d'années. Il a été défini pour décrire des architectures, pour les simuler ou les documenter, même si désormais on essaie de l'utiliser pour synthétiser des circuits. VHDL est un langage textuel qui permet de décrire des composants de différentes complexités, et à plusieurs niveaux, notamment RTL. Il est souvent utilisé comme interface entre divers outils.

### I.2.3.4 Niveau algorithmique

Le niveau algorithmique [23, 24] est le niveau le plus abstrait pour décrire un circuit. On peut spécifier le comportement d'une architecture à l'aide de graphes de dépendances (contrôle et/ou données). L'autre possibilité consiste à utiliser un des nombreux langages de description de matériel.

Les besoins spécifiques du matériel (contraintes de temps, de ressources, de performances) ont conduit à la définition de langages spécialisés pour la description d'architectures appelés HDL (*Hardware Description Language*). On peut classer les langages utilisés en deux styles : procédural et déclaratif. Il existe des passerelles pour aller de l'un à l'autre.

- **Langages procéduraux**

Ils précisent la procédure à suivre pour réaliser l'algorithme. Les HDL procéduraux reprennent ce découpage en blocs fonctionnels. Ils ont comme base un langage du type Pascal, C, ADA par exemple. On peut citer HardwareC, VHDL ou Verilog. Ces langages étant impératifs, une étape de compilation en un graphe de flot de données ou de contrôle est nécessaire pour faire apparaître le parallélisme.

- **Langages déclaratifs**

On peut citer Alpha, Signal, Silage. Il décrivent les relations entre les données, sans préciser un ordre explicite d'exécution d'une instruction. Ces langages sont souvent utilisés pour des domaines d'applications spécifiques (traitement du signal, architectures régulières, etc.) dont les algorithmes s'expriment naturellement de manière applicative.

### I.2.3.5 Niveau système

Le niveau de description le plus haut est le niveau système. Il spécifie un système de façon formelle. La traduction des fonctionnalités d'un système en une spécification fonctionnelle précise et sûre n'est pas aisée. On doit souvent passer par plusieurs étapes pour obtenir une spécification claire et correcte. Travailler à un niveau très abstrait permet justement de vérifier des fonctionnalités et de corriger des erreurs plus facilement. Au niveau le plus abstrait, la description consiste en un ensemble d'entités fonctionnelles et les relations qui existent entre elles. Différents modèles permettent de représenter un système :

- le modèle flot de données, DFG (*Data Flow Graph*), qui se prête bien aux applications de traitement du signal ;
- les machines d'états finis, FSM (*Finite State Machine*), souvent améliorées par la hiérarchie et la concurrence, qui décrivent bien les systèmes dominés par le contrôle ;
- le modèle des processus communicants, CSP (*Communicating Sequential Processes*), qui représente des systèmes complexes et parallèles et est donc plutôt utilisé pour générer du logiciel ;
- le modèle PSM (*Program-State Machine*), un FSM qui contient des instructions de programme, et qui a donc les mêmes visées que les modèles CSP et FSM.

Une fois le modèle choisi, la spécification fonctionnelle d'un système passe par l'utilisation d'un langage. L'adéquation entre le modèle et le langage peut-être plus ou moins bonne. Ainsi les langages relationnels sont plus adaptés au modèle flot de données. Alors que VHDL ou Verilog se prête plutôt à une modélisation de type CSP, voire FSM. SpecCharts supporte ces modèles et aussi le modèle PSM. Le langage SDL, quant à lui, correspond à un graphe de flot de données dont les feuilles sont des machines d'états finis.

Les outils d'aide à la conception sont nombreux et les langages variés. Les principaux outils de recherche développés sont les suivants.

- Ptolemy représente un système par un schéma de blocs interconnectés, et autorise plusieurs formalismes de spécification, notamment SDF, (*Synchronous Data Flow*), tout en restant orienté traitement du signal.
- le projet Cosmos utilise le langage SDL, mais travaille sur un format interne appelé SOLAR, qui est une machine d'états finis hiérarchiques modifiée.
- SpecSyn a comme langage SpecCharts.
- Vulcan prend une spécification en HardwareC (extension de C par des directives matériel) pour décrire des systèmes temps-réel.
- Cosyma propose plutôt une description ciblée logicielle en $C^x$ (également une extension de C), sous forme de processus communicants, et cible donc des systèmes complexes.

Ces outils permettent de décrire des systèmes mixtes (logiciel/matériel). La cible architecturale varie, suivant l'outil, sur le nombre d'ASICs, le nombre de processeurs généraux. La spécification système est modifiée pour obtenir un niveau algorithmique où chaque partie du système est ciblée sur un circuit donné. Durant tout le processus on peut simuler et vérifier l'application.
Certaines des idées développées dans ces systèmes ont été transférées vers des outils industriels tels que Cadence (SPW), Mentor Graphics, Synopsys (Cossap).

### I.2.4 Synthèse d'architectures

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

Le processus de synthèse peut contenir des étapes au sein d'un même domaine : la description est alors raffinée d'un premier niveau d'abstraction à un second niveau moins abstrait. Par exemple, on peut, au sein du domaine comportemental, aller du niveau algorithme au niveau transfert de registre. On parle également de synthèse pour désigner les étapes de transformations internes à un unique domaine.

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

### I.2.4.1 Nécessité de la synthèse architecturale

Aujourd'hui la technologie VLSI [25] permet d'implémenter des systèmes contenant plusieurs millions de transistors sur une simple puce. Pour exploiter cette technologie, les concepteurs ont besoin d'outils logiciels très évolués leur permettant de gérer ces transistors efficacement.

Au cours des dernières années, la synthèse logique a été reconnue comme partie intégrante du processus de conception, permettant ainsi de faire évoluer sa méthodologie du stade ``*placer-relier-simuler*`` à celui de `` *décrire-synthétiser*``. Cette nouvelle méthodologie permet de décrire un circuit de façon comportementale, d'éviter les détails d'implémentation et finalement de synthétiser sa structure à l'aide d'un outil de CAO.

Les concepteurs peuvent appliquer cette méthode à différents niveaux d'abstraction.

- Au niveau *porte*, ils peuvent synthétiser les unités fonctionnelles et de contrôles au moyen de synthèse logique combinatoire et les contrôleurs des machines à états au moyen de synthèse logique séquentielle.
- Au niveau *RTL*, ils peuvent décrire le comportement d'un ASIC avec des machines d'états généralisées dans lesquelles chaque état réalise des opérations arbitrairement complexes.
- Au niveau *architectural*, ils peuvent synthétiser, à partir de descriptions algorithmiques, des ASICS par le biais d'outils de synthèse de haut niveau. Cette méthode est de loin la plus rapide, car elle permet de décrire très simplement de façon comportementale des algorithmes qui seraient très compliqués au niveau *RTL*.

Cette synthèse permettrait donc à des non spécialistes du circuit, comme par exemple des personnes s'occupant de systèmes ou d'algorithmes (de traitement du signal), d'accéder facilement au silicium ou à des circuits programmables et ainsi de pouvoir évaluer la faisabilité matérielle (*Hardware*) de leurs algorithmes. D'autre part, dans le contexte économique actuel, les entreprises de conception de circuits doivent réduire au maximum le temps d'arrivée de leurs produits à la vente (*time to market*) tout en fiabilisant le cycle de développement. Ainsi, il devient de plus en plus important de porter l'aide logiciel le plus haut possible, c'est-à-dire au niveau de l'architecture des circuits.
Pour ce faire, de nombreux outils de synthèse d'architecture sont déjà en cours de conception mais ne sont encore, pour la plupart, que des travaux universitaires en cours de développement.

### I.2.4.2 Descriptions et contraintes

La synthèse architecturale peut se diviser en plusieurs grande étapes décrites dans les paragraphes suivants. La figure I.2.2 présente un synoptique général du flot d'une synthèse comportementale.

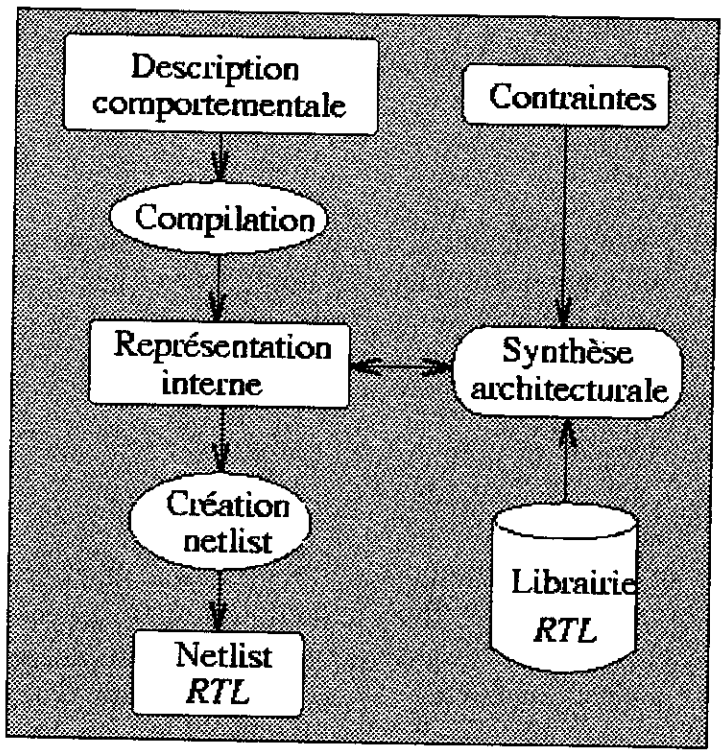



Figure I.2.2 : Flot de synthèse architecturale

### I.2.4.3 Les langage de description

La première étape est évidemment la création par le concepteur d'une description comportementale à priori indépendante de l'outil utilisé, mais qui tiendra compte des restrictions sémantiques propre à l'outil considéré. Cette description se fait dans un langage de haut niveau qui peut être, du VHDL, du HARDWAREC ou du VERILOG.
Il faut ensuite déterminer, en fonction de l'outil utilisé et des objectifs à atteindre, la ou les contraintes à mettre en œuvre lors de la synthèse. Certains outils peuvent travailler avec des contraintes temporelles, d'allocation, de surface ou encore de puissance.

Le VHDL est conforme à la norme IEEE et qu'il est très largement utilisé par les concepteurs, il représente le langage d'entrée de nombreux outils de synthèse logique. Il peut décrire le comportement des circuits à plusieurs niveaux d'abstraction (porte, *RTL*, architecture). IL accepte les assignations séquentiels et concurrentes, les boucles conditionnelles, les procédures et fonctions.
Le *Behavioral Compiler* de SYNOPSYS supporte du VHDL ou bien du VERILOG comme description de l'algorithme d'entrée. Le langage d'entrée sera donc du *VHDL comportmental* avec comme seule restriction le fait que cette description doit être mono-processus. Si la description comporte plusieurs processus, ils seront traités comme des circuits séparés. Ce langage, à la différence du VHDL logique synthétisable accepte la

clause *after* s'il s'agit d'un signal affecté par l'instruction *transport*, et si la durée est un multiple de l'horloge. Cette clause est alors interprétée comme un retard du signal en mode *cycle-fixed*.
Cette description utilise les fronts de l'horloge pour ordonnancer les calculs. Ainsi les seuls *Wait* utilisés seront des ``*Wait until Clock='1' and Clock'event*``. En fait, la seule difficulté d'utilisation de ce langage très complet est de définir les endroits où insérer les instructions *Wait* dans la description comportementale. Ce lieu dépend du mode d'ordonnancement des entrées-sorties choisi par le concepteur.

### I.2.4.4 Compilation

La description précédente est alors analysée (lexical, syntaxique) et compilée en une représentation interne sur laquelle l'outil va se baser pour effectuer la synthèse. Durant la compilation certaines transformations peuvent être opérée en vue d'optimiser la synthèse. Cette représentation est généralement sous forme d'un graphe flot de données avec ou sans noeud de contrôle.

### I.2.4.5 Sélection

Lors de la troisième étape, l'outil sélectionne les composants à utiliser dans le circuit. Ces composants sont choisis dans une librairie qui peut être désignée par l'utilisateur. Cette étape n'a évidemment de sens que si plusieurs opérateurs disponibles en bibliothèque sont en compétition pour une opération donnée. Cette étape a d'autant plus d'importance que la librairie RTL est complète et complexe.

### I.2.4.6 Allocation, ordonnancement et assignation

Puis vient l'étape la plus importante de la synthèse automatique, à savoir celle pendant laquelle les phases d'allocation, d'ordonnancement et d'assignation sont réalisées.

- L'étape d'allocation détermine le type et le nombre de ressources à utiliser dans l'architecture afin de respecter les contraintes données par le concepteur.
- Les opérations et les accès mémoires de l'algorithme sont alors ordonnancés par le logiciel.
- Enfin, la phase d'assignation relie chaque opération à une ou plusieurs ressources.

### I.2.4.7 Description de sortie

Finalement, une fois l'architecture créée, il ne reste plus qu'à la décrire dans le langage de sortie de l'outil considéré. Cette sortie est généralement décrite par une netlist au niveau *RTL* qui peut alors être synthétisée par un outil de synthèse logique.

## I.2.5 La méthodologie utilisée pour la conception sur composant FPGA

Notre approche de conception, déduite des considérations qui viennent d'être introduite et de l'outil de conception que nous avons utilisé, est illustrée par la figureI.2.3. Nous pouvons distinguer l'utilisation de deux types d'outils. Les outils spécifiques qui sont généralement fournis par les constructeurs de composants et les outils génériques qui sont quant à eux indépendants de la cible.

Au cours de l'étape de conception, le circuit est décrit en combinant les trois méthodes proposées par l'environnement de développement ***Xilinx Foundation Series 1.5, Student Version*** [26] :

- Le circuit global est saisi en schématique
- Certains blocs de fonctions sont décrit en langage VHDL, puis intégrés sous formes de macros dans le schéma global
- La logique de contrôle des différentes options architecturales de l'algorithme DES est décrite sous forme de machine d'états. Elle est ensuite insérée sous formes de macros dans le schéma global.

Chaque conception est testée dans son niveau fonctionnelle. Cette simulation fonctionnelle permet de tester la validité de notre conception. Aucune vérification temporelle n'est ici effectuée. Cette première phase de simulation nous a apporté un double intérêt. Elle nous a permis de valider fonctionnellement la description (entrée schématique, syntaxique ou en machine d'états) en s'affranchissant des étapes parfois longues de placement et de routage. Elle nous a permis aussi de valider (fonctionnellement) des sous – ensembles de la conception au fur et à mesure de l'avancement des travaux. Une seconde phase de simulation fonctionnelle a été réalisée après la phase de synthèse. Pour cette simulation fonctionnelle, nous avons utilisé l'outil ***Logic Simulator – Xilinx Foundation F1.5***.

La phase suivante est la synthèse du circuit qui consiste à mettre à plat toute la schématique (à l'aide de ***Schematic Compiler Sc version 3.1.19)***, à synthétiser les descriptions HDL comportementales (***Synopsys FPGA Macro Compiler version 3.1.1.0w)*** , à réduire les équations logiques et à les optimiser en fonction du composant cible. A ce stade de la conception, le fichier netlist est créé décrivant ainsi les interconnexions entre les éléments logiques du composant.

Le netlist et les fichiers de contraintes, générés par les outils de synthèses et par l'utilisateur spécifiant la période d'horloge maximum et les affectations des broches, sont utilisés par le logiciel Xilinx pour le placement et le routage de la logique sur un composant spécifique. Le résultat est la génération d'un bit-stream pour la programmation du composant. ***Le Xilinx Flow Engine version M 1.5.19*** a été utilisé pour cette étape.

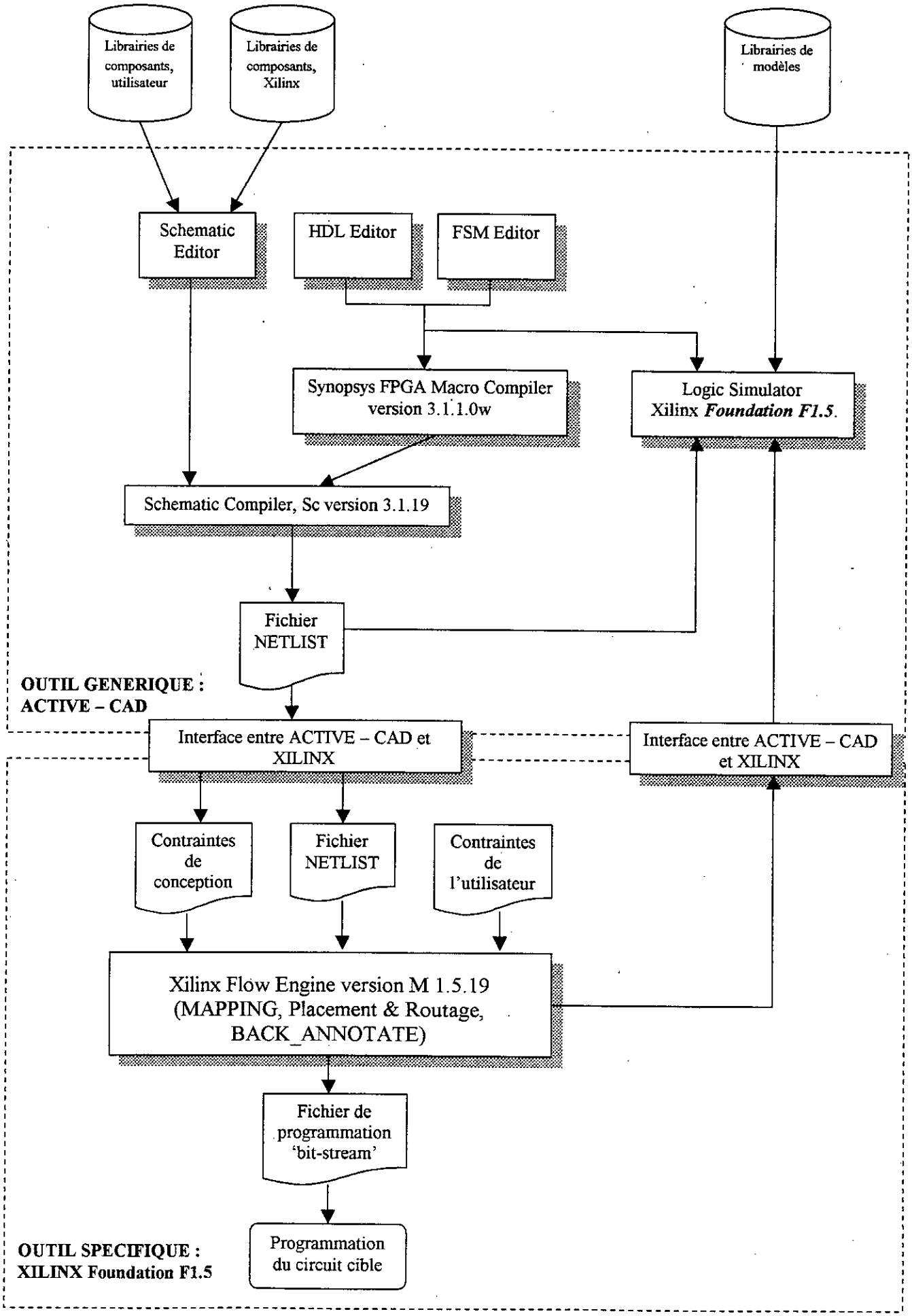



Figure I.2.3 : Les phases de conception

La phase finale est la vérification, une fois de plus, de notre conception. Cette fois-ci, il s'agit d'une vérification temporelle en utilisant le modèle de simulation généré par les outils Xilinx approriés. Ce modèle de simulation inclus le réseau d'interconnexions, les CLBs et les délais qui en y sont induits. Là aussi, nous avons utilisé l'outil ***Logic Simulator – Xilinx Foundation F1.5*** avec les mêmes vecteurs de test précédemment définis pour la simulation fonctionnelle.

### I.2.6 Conclusion

L'objectif qui nous a été assigné ; à savoir ; concevoir un processeur dédié se basant sur une méthode algorithmique donnée semble parfaitement épouser les modèles de conception actuelles exprimés à travers des environnements de développement orientés déduits, d'une manière générale, du diagramme en Y introduit par Gajski. L'outil CAO utilisé déduit de la cible technologique préalablement choisie nous amène à justifier et à introduire au préalable le modèle algorithmique choisi.

Le fait d'avoir dans la structure interne du cryptoprocesseur, un bloc de chiffrement basé sur un algorithme cryptographique exige la connaissance préalable de certaines bases théoriques de la cryptographie. Nous citons :

- les constituants d'un cryptosystème et ses différents modes de fonctionnement,
- les techniques et les algorithmes cryptographiques,
- le chiffrement logiciel versus le chiffrement matériel,
- l'utilisation des avantages apportés d'une part par les algorithmes à clefs secrètes et d'autre part par les algorithmes à clefs publiques, pour la construction d'un cryptosystème mixte réalisant les fonctions cryptographiques de chiffrement de données et d'échange de clefs.

Tous ces fondements cryptographiques seront abordés dans le sous-chapitre I.3.

# I.3 - Concepts fondamentaux de la cryptographie

## I.3.1 Introduction

Ce chapitre a pour but essentiel de montrer quels sont les constituants des cryptosystèmes ou systèmes de cryptage, de donner leurs principes de fonctionnement et les concepts à définir pour leurs réalisations. Il justifie nos différents choix tel que :

- le choix d'une implémentation matérielle au lieu d'une l'implémentation logicielle,
- le choix d'un cryptosystème mixte assurant les fonctions cryptographiques de chiffrement de données et d'échange de clefs.

## I.3.2 Définition de la cryptographie

La cryptographie est la science qui permet de rendre une information inaccessible sauf, pour ceux à qui elle est destinée. La cryptographie existe depuis des millénaires. Elle a débuté avec un simple système de permutation de lettres et de chiffres accompagné d'un code. Que le code ou la clef soit élaboré manuellement ou par ordinateur, le chiffrement revient à mettre en œuvre une fonction mathématique appliquée à l'information cible. La même fonction mathématique ou une fonction connexe est utilisée pour réaliser la fonction inverse.

## I.3.3 Définition de la cryptanalyse

Comme on vient de le définir, le but de la cryptographie est de préserver le texte en clair de l'indiscrétion des attaquants (aussi appelés adversaires, opposants, cryptanalyses, etc). contrairement à la cryptanalyse qui correspond à la science de la reconstitution du texte en clair sans connaître la clef. Une cryptanalyse réussie peut fournir soit le texte en clair, soit la clef.
Une attaque est basée sur l'hypothèse que le cryptanalyste dispose de la connaissance complète de l'algorithme de chiffrement. Bien que cela ne soit pas toujours le cas dans le monde réel de la cryptanalyse, cette hypothèse est adoptée dans le monde académique de la cryptanalyse.
Il y a quatre types génériques d'attaques cryptanalytiques reprises ci-dessous dans l'ordre croissant d'efficacité [2].

- L'attaque à texte chiffré connu.
  L'opposant ne connaît que la chaîne du message chiffré **Y**.

- L'attaque à texte en clair connu.
  L'opposant dispose d'un texte clair **X** et du texte chiffré **Y** correspondant.

- L'attaque à texte en clair choisi.
  L'opposant a accès à une machine chiffrante. Ainsi, il peut choisir un texte clair **X** et obtenir son texte chiffré **Y**.

- L'attaque à texte chiffré choisi.
  L'opposant a temporairement accès à une chaîne déchiffrante. Ainsi, il peut choisir un texte chiffré **Y** et obtenir son texte clair **X** correspondant.

### I.3.4 Définition d'un cryptosystème

On définit un cryptosystème comme étant un quintuplet *(P, C, K, E, D)* satisfaisant :

1. $P$ est un ensemble fini de messages de ***textes clairs*** possibles
2. $C$ est un ensemble fini de messages de ***textes chiffrés*** possibles
3. $K$ est un ensemble fini de ***clefs*** possibles
4. Pour tout $k \in K$, il y a une ***règle de chiffrement*** $e_k \in E$ et une ***règle de déchiffrement*** correspondante $d_k \in D$. Chaque $e_k : P \to C$ et $d_k : C \to P$ sont des fonctions telles que $d_k(e_k(x)) = x$ pour tout texte clair $x \in P$.

Ce cryptosystème est illustré par la Figure I.3.1. Il va permettre à deux personnes, appelées traditionnellement Alice et Bob, de communiquer à travers un canal peu sûr de telle sorte qu'un opposant, Oscar, ne puisse comprendre ce qui est échangé.

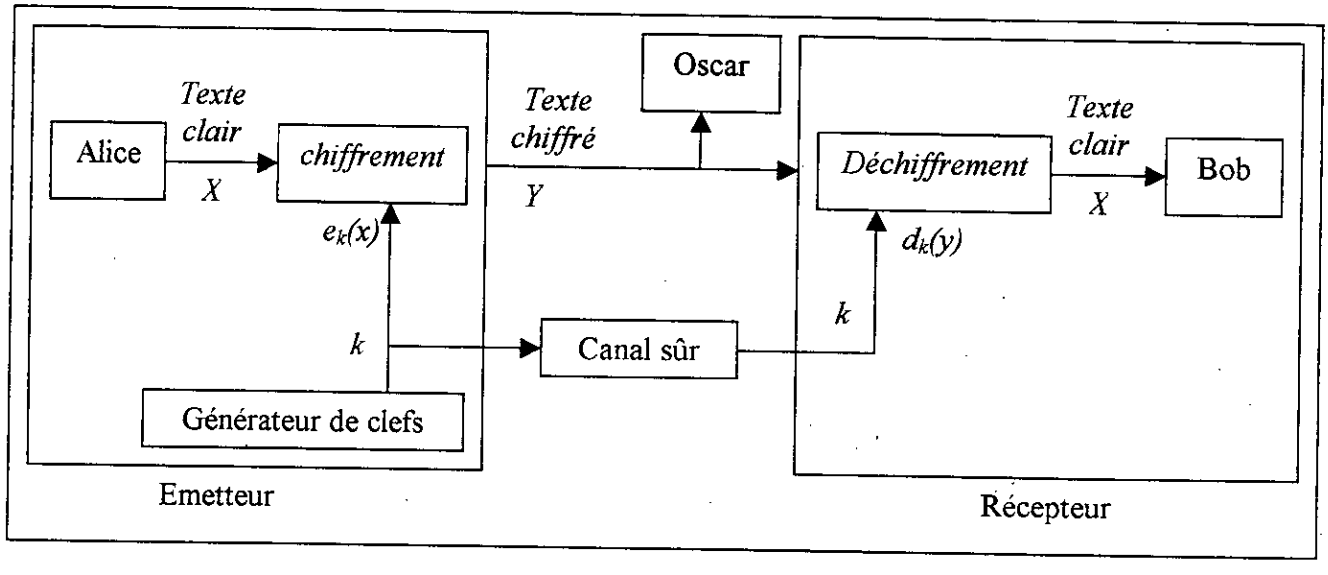



Figure I.3.1 : Schéma général de la communication chiffrée entre un émetteur et un récepteur

Dans le jargon cryptographique et pour illustrer certaines opérations, on engage des acteurs. La liste de ces acteurs et leurs rôles respectifs est donné par le tableau ci-dessous.

| | |
|---|---|
| Alice | participante de toutes les opérations |
| Bob | participant de toutes les opérations |
| Carol | participante des opérations à trois ou quatre participants |
| Dave | participant des opérations à quatre participants |
| Eve | espionne |
| Oscar | attaquant malveillant actif |
| Trent | arbitre intègre |
| Warden | gardien : il surveillera Alice et Bob dans certains protocoles |
| Peggy | plaideuse |
| Victor | vérificateur (contrôleur) |

Tableau I.3.1 : Liste des acteurs et de leur rôle

### I.3.4.1 Typologie des cryptosystèmes

Avant l'avènement des ordinateurs, la cryptographie traitait des systèmes basés sur des lettres (ou caractères). Les différents algorithmes cryptographiques remplaçaient des caractères par d'autres (*chiffrement par substitution*) ou transposaient les caractères (*chiffrement par permutation*). Les meilleurs systèmes faisaient les deux opérations plusieurs fois, c'est plus complexe de nos jours, mais la philosophie est restée essentiellement la même. La différence majeure est que les algorithmes actuels manipulent les bits au lieu de caractères. Ce n'est finalement qu'un changement de taille d'alphabet: on passe de 26 éléments à 2 éléments.

Les nouvelles tendances pour la conception des systèmes cryptographiques sont basées sur l'idée de la combinaison des systèmes cryptographiques en formant leur "produit" tel que c'est le cas du *Standard de Chiffrement de Données DES.*

Les cryptosystèmes qui sont cités ci-dessous décrivent brièvement le fonctionnement des trois catégories de systèmes cryptographiques : cryptosystèmes par substitution, cryptosystèmes par permutation et les cryptosystèmes produit.

#### a) Cryptosystèmes par substitution

Un chiffre à substitution est un chiffre dans le quel chaque caractère du texte en clair est remplacé par un autre caractère dans le texte chiffré. Ce système est défini dans la figure I.3.2.

Soit $P = C = Z_{26}$ où $Z_{26}$ est l'ensemble des 26 lettres de l'alphabet.
$K$ est l'ensemble des permutations sur l'ensemble des 26 nombres 0, 1, ...,25. Pour chaque permutation $\pi \in K$, on définit

$$e_\pi = \pi(x)$$

et

$$d_\pi(y) = \pi^{-1}(y)$$

où $\pi^{-1}$ est la permutation réciproque de $\pi$

Figure I.3.2 : Chiffrement par substitution

**b) Cryptosystèmes par permutation**

Un chiffre à permutation est un chiffre dans lequel les caractères du texte en clair demeurent inchangés mais dont les positions respectives sont modifiées. Une définition est présentée sur la figure I.3.3.

Soit un entier strictement positif $m$. soit $P = C = \{0,1,...,25\}^m$ et soit $K$ l'ensemble des permutations de $\{1,...,m\}$. Pour toute clef $\pi$ (c'est-à-dire pour toute permutation), on définit

$$e_\pi(x_1,...,x_m) = (x_{\pi(1)},...,x_{\pi(m)})$$

et

$$d_\pi(y_1,...,y_m) = (y_{\pi^{-1}(1)},...,y_{\pi^{-1}(m)})$$

où $\pi^{-1}$ est la permutation inverse de $\pi$

Figure I.3.3 : Chiffrement par permutation

**c) Cryptosystèmes produit**

Une notion introduite par Shannon en 1949 est l'idée de combiner des systèmes cryptographiques en formant leur "produit". Cette idée joue un rôle fondamental dans la conception des systèmes cryptographiques actuels tels le **Standard de Chiffrement de Données DES**.

Pour simplifier, on se contentera d'étudier les systèmes cryptographiques dans lesquels $C = P$. Nous les appellerons des systèmes endomorphiques. Soit $\boldsymbol{S}_1 = (P, P, K_1, E_1, D_1)$ et $\boldsymbol{S}_2 = (P, P, K_2, E_2, D_2)$ deux systèmes cryptographiques endomorphiques sur le même espace de textes clairs (et de textes chiffrés). Le produit $\boldsymbol{S}_1 \times \boldsymbol{S}_2$ de $\boldsymbol{S}_1$ et $\boldsymbol{S}_2$ se définit par un système cryptographique.

$$(P, P, K_1 \times K_2, E, D)$$

une clef du produit des systèmes est de la forme $k = (k_1, k_2)$ où $k_1 \in K_1$ et $k_2 \in K_2$. les règles de chiffrement et de déchiffrement de ce système se définissent ainsi : pour chaque $k = (k_1, k_2)$, on a une règle $e_k$ définie par

$$e_{(k1,k2)}(x) = e_{k2}(e_{k1}(x))$$

et une règle de déchiffrement $d_k$ définie par

$$d_{(k1,k2)}(y) = d_{k1}(d_{k2}(y))$$

autrement dit, on chiffre $x$ d'abord avec $e_{k1}$ et l'on "surchiffre" le résultat avec $e_{k2}$. Le déchiffrement est semblable, mais dans l'ordre inverse :

$$\begin{aligned} d_{(k1,k2)}(E_{(K1,k2)}(x)) &= d_{(k1,k2)}(e_{k2}(e_{k1}(x))) \\ &= d_{k1}(d_{k2}(e_{k2}(e_{k1}(x)))) \\ &= d_{k1}(e_{k1}(x)) \\ &= x. \end{aligned}$$

Si l'on prend le produit d'un système cryptographique endomorphique **S** avec lui-même, on obtient le système $\boldsymbol{S} \times \boldsymbol{S}$ que l'on note $\boldsymbol{S}^2$. Si l'on fait le produit de n systèmes identiques $\boldsymbol{S}$, on obtient un système $\boldsymbol{S}^n$. On l'appelle le *système cryptographique itéré*.

Un système cryptographique $\boldsymbol{S}$ est dit *idempotent* si $\boldsymbol{S}^2 = \boldsymbol{S}$. Les deux systèmes cryptographiques étudiés à savoir le **chiffrement par substitution** et le **chiffrement par permutation** sont tous idempotents. Bien sûr, si un système est idempotent, on n'a aucun intérêt à utiliser ses systèmes itérés, car ils nécessitent plus de clefs mais n'apportent aucune sécurité supplémentaire.

Si un système cryptographique n'est pas idempotent, la sécurité est notoirement augmentée en l'itérant plusieurs fois. Cette idée est utilisée dans le **Standard de Chiffrement de Données DES**, qui utilise 16 itérations.

### I.3.4.2 Conception d'un cryptosystème

Pour réaliser ce cryptosystème, on doit définir impérativement les trois concepts complémentaires suivants :

- le protocole cryptographique,
- les techniques cryptographiques,
- l'algorithme cryptographique.

### I.3.4.2.1 Le protocole cryptographique

Un protocole cryptographique est une série d'étapes, impliquant deux ou plusieurs participants, conçu pour accomplir une tâche. Lors de l'exécution de ses différentes étapes, des algorithmes cryptographiques sont mis en œuvre.

Dans tout réseau de communication, il existe des utilisateurs malhonnêtes. Pour s'y prémunir, les protocoles cryptographiques ont été formalisés afin d'examiner comment des participants malhonnêtes peuvent essayer de tricher et d'en déduire des protocoles qui les excluent.

Les différents types de protocoles cryptographiques peuvent être classés comme suit :

- ***protocoles arbitrés***

  Un **arbitre** est un tiers, personne de confiance et désintéressée, qui veille au bon déroulement du protocole (voir figure I.3.4a). « Désintéressée » parce qu'il n'a pas d'intérêt dans l'accomplissement du protocole. Les arbitres aident au bon accomplissement de protocoles entre participants méfiants.

- ***Protocoles avec juge – arbitre***

  Les protocoles arbitrés peuvent être divisés en deux sous - protocoles : un sous protocole non arbitré qui est exécuté chaque fois que le protocole est utilisé et un autre sous – protocole arbitré auquel on ne fait appel que s'il y a un litige entre les participants. Dans ce cas particulier, on ne parlera pas d'arbitre mais **de juge – arbitre** (voir figure I.3.4b).

- ***Protocoles à discipline intrinsèque***

  Ce type de protocole garantit lui-même l'intégrité de la transaction (voir figure I.3.4c). Si l'un des participants essaie de tricher, l'autre participant le détecte immédiatement et le protocole est alors interrompu.

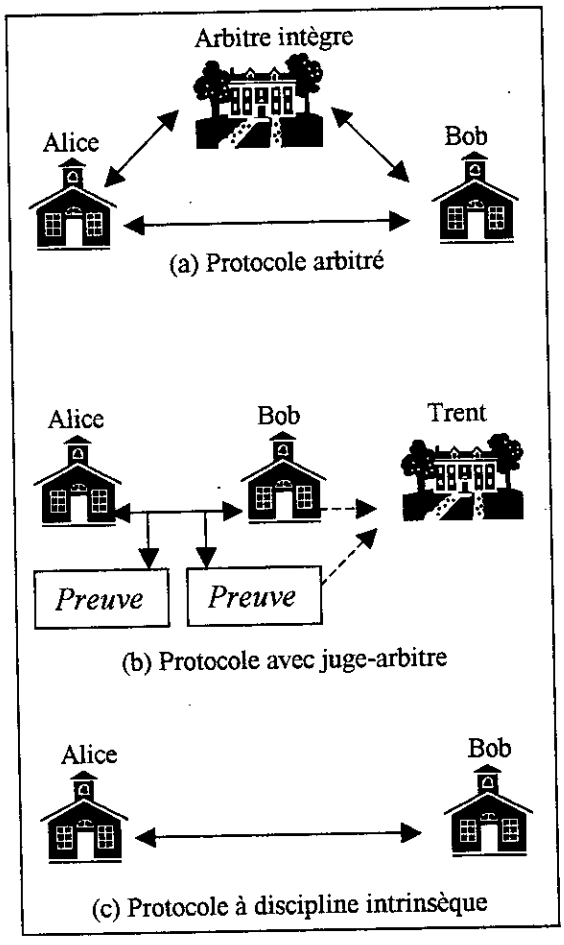



Figure I.3.4 : Types de protocoles

- **Exemples de protocoles cryptographiques selon le types de messages échangés**

Le choix du protocole cryptographique dépend du type de message que l'on veut échanger. Ainsi le but d'un protocole va au delà de la simple confidentialité. Les participants d'un protocole cryptographique peuvent vouloir partager partiellement un secret pour calculer une valeur, engendrer une suite aléatoire, convaincre l'autre interlocuteur de son identité ou signer simultanément un contrat.

Par conséquent, plusieurs types de protocoles cryptographiques [2] ont été développés, on cite :

- les protocoles de communications à l'aide de cryptosystèmes,
- les protocoles de signatures numériques,
- les protocoles d'échange de clefs,
- les protocoles d'authentification.

### I.3.4.2.2 Les techniques cryptographiques

Cette partie concerne les techniques à mettre en œuvre pour traiter des contraintes qui sont liés aux constituants d'un cyptosystème :

- les clefs : quelle doit être leurs longueurs.
- leur gestion : comment les engendrer , les stocker, les détruire et autres.
  Cette fonction est considérée comme la partie la plus difficile de la cryptographie. Elle doit être très sûre.
- la mise en œuvre de la cryptographie. Comment choisir les algorithmes pour l'échange des données de base, leur stockage ou autre.

#### a) Les clefs

La sécurité d'un cryptosystème dépend de deux critères : la solidité de l'algorithme et la longueur de la clef [1]. Si nous supposons que la solidité d'un cryptosystème est basée uniquement sur l'hypothèse que l'attaquant ne connaît pas les détails de l'algorithme utilisé, ce cryptosystème sera alors très vite cassé par le simple fait de désassembler le code qui lui est associé et (ou) de remonter à l'algorithme générateur.
Cela nous amène à formuler une remarque très importante : la sécurité d'un cryptosystème doit résider dans la clef et non pas dans les détails de l'algorithme.

#### a.1) Longueur de la clef

Si nous supposons qu'un cryptosystème présente une solidité parfaite; le meilleur moyen d'essayer de le casser revient à essayer toutes les clefs possibles. On parlera alors d'**attaque exhaustive**.
Si la clef a une longueur de 8 bits; il y aura $2^8 = 256$ clefs possibles. Il faudra donc 256 tentatives pour trouver la bonne clef et en moyenne 128 suffiront.
Si la clef a une longueur de 64 bits, on disposera alors de $2^{64} \approx 1{,}8 \times 10^{19}$ clefs possibles. Cette dernière longueur engendrera sur un ordinateur pouvant réaliser le test d'un million de clefs par seconde un délai de 600 000 ans pour trouver la bonne clef.

*Une question* est alors posée : qu'elle doit être la longueur optimale d'une clef ?
***La réponse*** dépend de l'application, de la durée de confidentialité des données et de leur(s) importance(s).
Nous pouvons même spécifier nos desiderata de sécurité. Par exemple :

- imposer une longueur de clef de telle sorte qu'il n'y ait pas une probabilité de plus de 1 pour $2^{32}$ pour casser le cryptosystème,
- imposer une somme critique à dépenser par un attaquant pour casser le système en moins d'un an,
- tenir compte du taux annuel de croissance de la performance des équipements de calcul sur la période totale de la recherche de la clef.

Des études [27] montrent que le rapport entre l'efficacité des équipements de calcul et le prix augmente d'un facteur 10 tous les 5 ans.

Le tableau I.3.2 donne une idée du niveau de sécurité requis pour différents types d'informations.

| Type de trafic | Longévité | Longueur minimale de la clef |
|---|---|---|
| Informations militaires tactiques | Minutes/heures | 56 bits |
| Secrets commerciaux | Décades | 64 bits |
| Secrets de la bombe H | > 40 ans | 128 bits |
| Identité des espions | > 50 ans | 128 ans |
| Embarras diplomatiques | > 65 ans | Au moins 128 bits |

Tableau I.3.2 : Niveau de sécurité nécessaire pour les différents types d'informations.

**b) Gestion des clefs**

Il ne serait d'aucune utilité de prendre toutes les marges de sécurité concernant la longueur des clefs alors que rien n'a été fait pour que ces clefs restent secrètes. Aussi, un point très sensible réside dans la gestion des clefs.

En pratique, la gestion des clefs est le problème de sécurité le plus difficile. Concevoir des algorithmes et des protocoles sûrs n'est pas chose aisée mais être sûr que les clefs restent secrètes est encore plus difficile. Les cryptanalystes attaquent souvent les algorithmes à clef secrète et les algorithmes à clef publique à travers leur système de gestion de clefs.
Pourquoi s'évertuer à casser l'algorithme cryptographique quand il est plus facile de retrouver les clefs à cause d'un relâchement dans les procédures de gestion des clefs ?

**b.1) Génération des clefs**

Si l'on utilise un processus « cryptographiquement » faible pour engendrer les clefs, alors tout le système est faible malgré l'utilisation d'algorithme de chiffrement le plus sûr qui puisse exister. L'attaquant n'aurait pas à essayer de cryptanalyser l'algorithme de chiffrement quand il peut cryptanalyser l'algorithme de génération de clefs.
Les faiblesses d'un algorithme de génération de clefs peuvent être dues :

- ***Cardinalité de l'ensemble de clefs***

Si le logiciel de chiffrement impose certaines contraintes sur l'utilisation des chaînes de caractères; ceci réduit considérablement le nombre de clefs possibles.
En prenant comme exemple l'algorithme de chiffrement **DES** disposant une clef de 56 bits de longueur et si toute chaîne de 56 bits peut être la clef; il y aura $2^{56}$ ($\approx 10^{16}$)

clefs possibles. Si par contre l'algorithme **DES** n'autorise que des clefs faites de lettres minuscules et de chiffres ; il n'autoriserait alors que $10^{12}$ clefs possibles.

Le tableau I.3.3 donne le nombre de clefs possibles en fonction de certaines contraintes sur les chaînes de caractères à l'entrée.

| | 4 octets | 6 octets | 8 octets |
|---|---|---|---|
| Lettres minuscules (26) | 460 000 | $3,1x10^{8}$ | $2,1x10^{11}$ |
| Lettres minuscules et chiffres (36) | 1 700 000 | $2,2x10^{9}$ | $2,8x10^{12}$ |
| Caractères alphanumériques (62) | $1,5x10^{7}$ | $5,7x10^{10}$ | $2,2x10^{14}$ |
| Caractères affichables (95) | $8,1x10^{7}$ | $7,4x10^{11}$ | $6,6x10^{15}$ |
| Caractères ASCII (128) | $2,7x10^{8}$ | $4,4x10^{12}$ | $7,2x10^{16}$ |
| Caractères ASCII 8 bits (256) | $4,3x10^{9}$ | $2,8x10^{14}$ | $1,8x10^{19}$ |

Tableau I.3.3 : Nombre de clefs possibles pour les différent espaces de clefs

- ***Mauvais choix de clefs***

Il faut choisir une clef de telle façon à éviter une attaque par dictionnaire [2]. Cette dernière consiste à mener une attaque en essayant les mots, les noms propres, les extensions triviales, etc.

- ***Génération de bonnes clefs***

Pour réussir à avoir de bonnes clefs, il faut :

- qu'elles soient formées de chaînes aléatoires de bits.
- engendrer les bits de la clef à partir d'une source aléatoire fiable ou d'un générateur pseudo-aléatoire de bits cryptographiquement sûr.
- éliminer les clefs faibles générées par certains algorithmes cryptographiques qui sont moins sûres que les autres clefs.
- utiliser la technique appelée **broyage de clef** [1] qui transforme des textes facile à mémoriser en des clefs aléatoires. Utiliser une fonction de hachage [2] à sens unique pour transformer un texte de longueur quelconque en une chaîne de bits pseudo-aléatoire.
- que les germes [1] de ces générateurs de clefs doivent être aléatoires.

**Exemple** : *Génération de clefs suivant le standard ANSI X9.17*

Le standard **ANSI X9.17** désigne une méthode de génération de clefs. Ce processus est adapté à la génération de clefs de session. L'algorithme cryptographique

utilisé pour engendrer les clefs est le **DES** mais pourrait être aisément tout autre algorithme.
Soit $E_k(X)$ le chiffrement **DES** de $X$ avec la clef $k$. La clef $k$ est la clef réservée à la génération de clefs. $V_0$ est un germe de 64 bits secret. T est une datation. Pour engendrer la clef aléatoire $R_i$, on calcule :

$$R_i = E_k(E_k(T_i) \oplus V_i).$$

Pour engendrer $V_{i+1}$, on calcule :

$$V_{i+1} = E_k(E_k(T_i) \oplus R_i).$$

Pour transformer $R_i$ en une clef **DES**, on ajuste simplement un bit sur huit pour la parité. Si on a besoin d'une clef de 128 bits, on doit engendrer une paire de clefs jointes.

### b.2) Transfert de clefs

Plusieurs approches essayent de résoudre ce type de problème :

- Une première solution serait de morceler la clef et d'envoyer par le suite chaque morceau sur un canal différent (voir la figure I.3.5). Un adversaire sera capable de reconstruire une clef que s'il arrive pas à collecter tous les morceaux transmis.

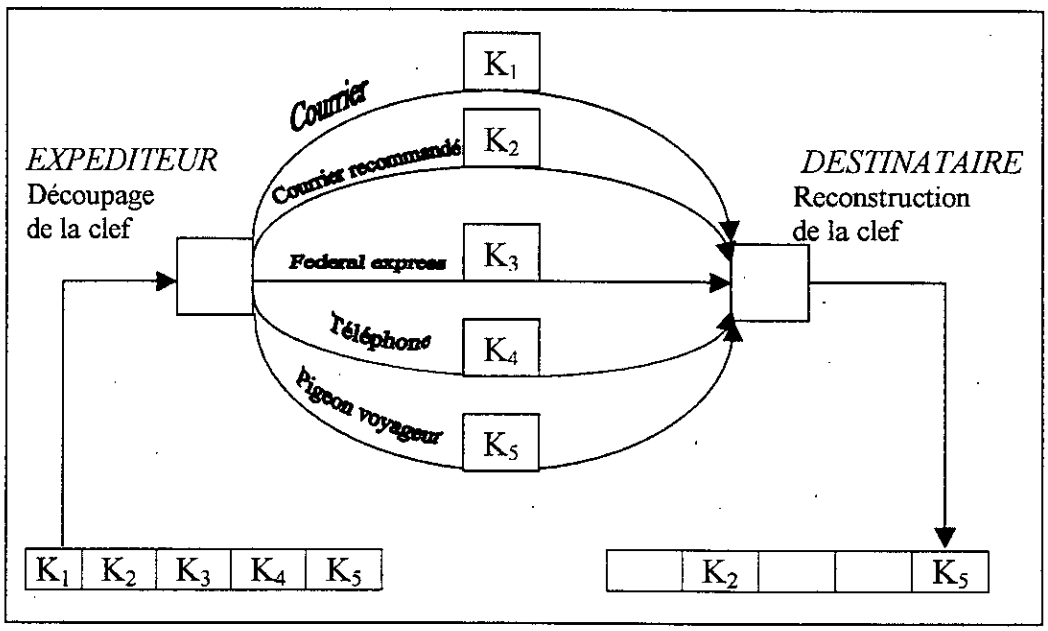



Figure I.3.5 : Distribution d'une clef morcelée

L'inconvénient majeur de cette solution peut apparaître dans le cas d'un changement fréquent d'une clef. Une solution pour les utilisateurs de cette technique consiste à transférer en une seule fois en un mois toutes les clefs. Cette liste de clefs a beaucoup de valeur et doit donc être rangée d'une manière très sûre après réception.

- Une deuxième solution, consiste à créer une **clef maîtresse** ou une **clef de chiffrement de clefs**. Cette clef est envoyée de manière sûre; soit par une rencontre entre personne ou soit par la technique de morcellement.

**b.3) Stockage des clefs**

Plusieurs approches ont été développées :

- Certains systèmes prennent une approche facile. L'utilisateur doit mémoriser la clef et ne doit jamais la stocker dans le système, il l'introduire chaque fois qu'un message doit être chiffré ou déchiffré.

- Une autre solution consiste à morceler d'abord la clef en deux moitiés et de stocker par la suite les deux moitiés respectivement dans le terminal et dans une clef "ROM" ou sur la piste d'une carte magnétique. Les utilisateurs peuvent alors introduire leur clef dans le système en insérant la clef ou la carte dans un lecteur spécial attaché à la boîte de chiffrement ou au terminal informatique.

- Des clefs difficiles à mémoriser peuvent être stockées sous forme chiffrée en utilisant des méthodes similaires à celles d'une clef de chiffrement de clefs.

**b.4) Longévité des clefs**

Aucune clef de chiffrement ne doit être utilisée pour une période indéfinie. Il y a plusieurs raisons à cela :

- Plus longtemps la clef est utilisée, plus elle a de chances d'être compromise.

- Plus longtemps la clef est utilisée, plus grande est la perte si la clef est compromise.

- Plus longtemps la clef est utilisée, plus grande est la tentation pour quelqu'un de fournir l'effort nécessaire pour la retrouver même si cet effort est une attaque exhaustive.

La nature de l'application cryptographique, détermine la longévité d'une clef :

- les clefs de session de communication doivent avoir des longévités relativement courtes en fonction de la valeur des données transmises et du volume de données chiffrées durant une période déterminée. Les clefs de session doivent être changées au minimum chaque jour.

- Les clefs de chiffrement de clefs ne doivent pas être changées aussi souvent. Elles sont utilisées occasionnellement pour l'échange de clefs. Dans

certaines applications, elles sont changées une fois par mois ou une fois par an.

- Les clefs de chiffrement utilisées pour chiffrer des fichiers archivés ne doivent pas être changées souvent. Une manière pour augmenter le degré de sécurité serait de chiffrer chaque fichier avec une clef unique par fichier et ensuite de chiffrer toutes les clefs avec une clef de chiffrement de clefs.

- Les clefs privées de cryptographie à clef publique utilisées pour les signatures numériques et les preuves d'identité doivent durer des années (voir même une vie entière). Pour d'autres applications, les clefs privées peuvent être ignorées directement après avoir terminé le protocole ou bien encore, les laisser valides que pendant un certain nombre d'années.

### c) Utilisation des algorithmes

Les principaux types de cryptosystèmes utilisés aujourd'hui se répartissent en deux grandes catégories : les cryptosystèmes par flots et les cryptosystèmes par blocs. Nous illustrons ci-après les différences fondamentales.

### c.1) Mode de chiffrement par blocs

Dans ce mode de chiffrement, le texte clair est fractionné en blocs de même longueur à l'aide d'une clef unique $k$. Emetteur et récepteur disposent de deux algorithmes de chiffrement/déchiffrement parfaitement réciproques, paramétrés par $k$ ( figure I.3.6).
Les caractéristiques de ces systèmes sont en général liées à leur très forte sensibilité à la dépendance inter-symboles [28]. En clair, si $X_n$ et $X_n'$ sont deux messages «voisins», il faut que les cryptogrammes associés $Y_n$ et $Y_n'$ soient non corrélables.

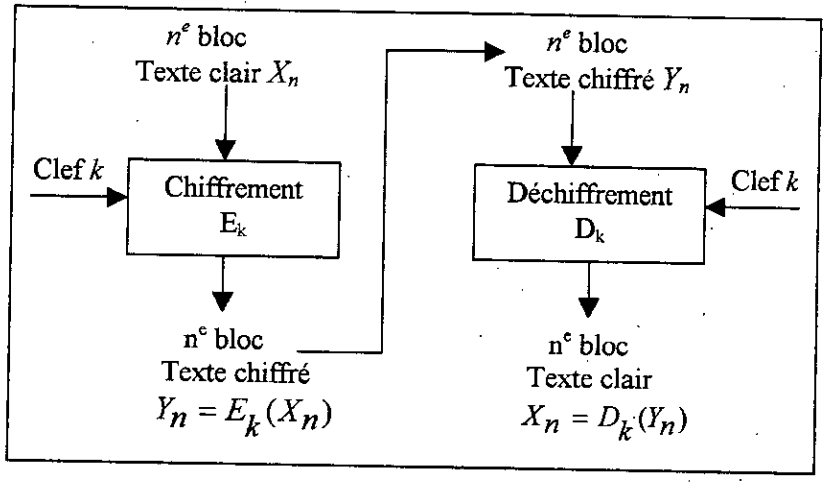



Figure I.3.6 : Système de chiffrement par blocs

### c.2) Mode de chiffrement par flot

Le chiffrement des messages se fait caractère par caractère au moyen de substitutions.
En d'autres termes, la lettre $X_n$ du message à transmettre est chiffrée en $Y_n = X_n + k_n$ (figure I.3.7).
Dans la pratique, les valeurs aléatoires des $k_i$ sont générées par un processus déterministe simulant une loi uniforme [29]. Ce processus est un algorithme dépendant d'un vecteur d'initialisation Z.

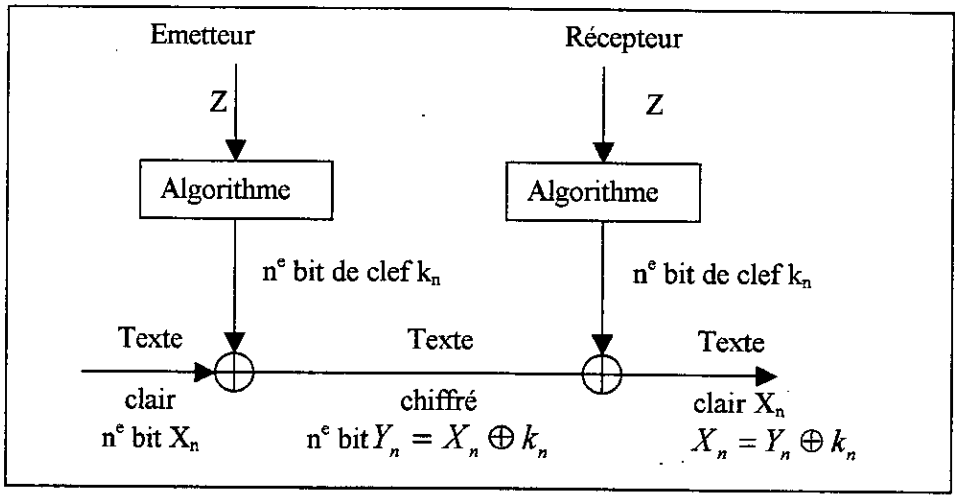



Figure I.3.7 : Système de chiffrement par flots

***Chiffrement par blocs versus chiffrement en continu***

- Les systèmes de chiffrement en continu ne chiffrent et déchiffrent qu'un bit à la fois. Ils ne sont pas vraiment adaptés à des réalisations logicielles. Les systèmes de chiffrement par blocs sont nettement plus faciles à implémenter par logiciel car les manipulations de bits sont coûteuses en temps de calcul.

- Cryptographiquement parlant, les systèmes de chiffrement par blocs sont plus généraux et leurs algorithmes peuvent être, en principe, rendus plus sûrs. D'un autre côté, les systèmes de chiffrement en continu sont plus faciles à analyser mathématiquement.

- Une autre différence essentielle entre les deux est la propagation d'erreurs. Avec le chiffrement par blocs, une seule erreur corrompra au moins un bloc de données. Avec certains modes de rétroaction, cela peut même être pire. Le chiffrement en continu peut être réalisé de telle manière qu'un seul bit de texte chiffré corrompu engendre un seul bit de texte corrompu. Donc tout dépend des applications et des taux de propagation d'erreurs afin que celles-ci soient détectées et éliminées.

- Le choix dépend aussi du type de données chiffrées.

*Chiffrement matériel et chiffrement logiciel*

Bien que le chiffrement par logiciel prenne de l'importance aujourd'hui, les raisons de la prédominance des réalisations matérielles sont :

*1. La Vitesse*

Le chiffrement est une tâche de calcul intensif. Mobiliser le processeur d'un ordinateur rien que pour cette tâche reste inefficace. Pour cette raison, confier le chiffrement à un circuit dédié rend tout le système plus rapide.

*2. La sécurité*

Manque de protection physique pour les algorithmes réalisés en logiciel. Un attaquant peut utiliser différents outils de "déverminage" et subrepticement modifier l'algorithme sans que qui que ce soit s'en rende compte. Les dispositifs de chiffrement matériels peuvent être bien encapsulés pour prévenir cela :

- Des boîtes à l'épreuve de l'investigation peuvent empêcher que quelqu'un modifie le dispositif de chiffrement matériel.

- Des puces VLSI dédiées peuvent être recouvertes d'un film protecteur chimique spécial de sorte que toute tentative d'accéder à l'intérieur provoque la destruction des circuits logiques de la puce.

- Des "boîtes" dédiées au chiffrement peuvent être blindées de manière à ne pas laisser échapper de rayonnement électromagnétique, vecteur potentiel de transmission involontaire d'informations.

**I.3.4.2.3 L'Algorithme cryptographique**

Un algorithme cryptographique est une fonction mathématique utilisée pour le chiffrement et le déchiffrement. Il y a deux principaux types d'algorithmes à base de clefs :

**a)** ***Les algorithmes à clefs secrètes***

Illustrés par figure I.3.8; ce sont des algorithmes où la clef de chiffrement peut être calculée à partir de la clef de déchiffrement ou vice versa. Dans la plupart des cas, la clef de chiffrement et la clef de déchiffrement sont identiques. Pour de tels algorithmes, l'émetteur et le destinataire doivent se mettre d'accord sur une clef à utiliser avant d'échanger des messages. La sécurité repose sur la clef.

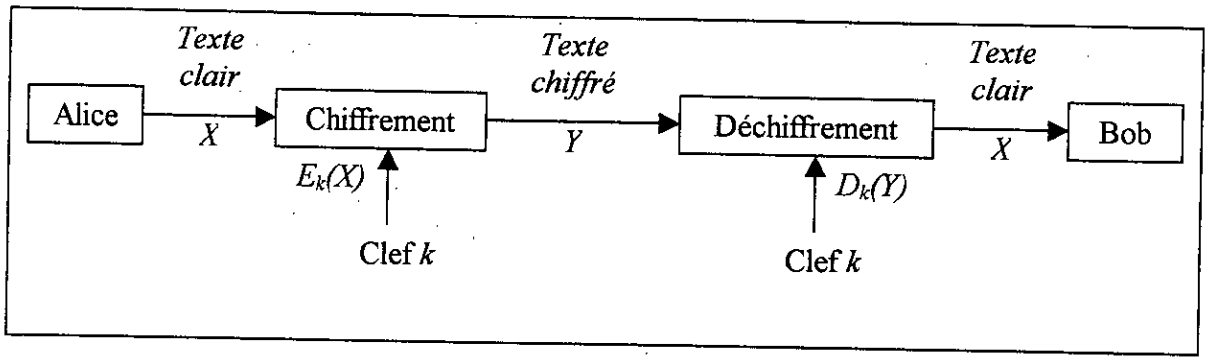



Figure I.3.8 : Chiffrement et déchiffrement avec un algorithme à clef secrète

**b)** ***Les algorithmes à clef publique***

Son principe fondamental consiste à utiliser des clefs de chiffrement et de déchiffrement différentes non reconstructibles l'une à partir de l'autre. Les avantages fondamentaux de ce type de cryptosystème sont les suivants :

- Si $K_{pr}$ et $K_{pu}$ sont les deux clefs possédées par un utilisateur, l'une peut être rendue publique, et l'autre privée ; d'où un premier avantage, celui de la ***confidentialité***. Connaissant la clef publique de l'utilisateur A, $K_{PuA}$ par exemple, chacun peut chiffrer sous cette clef, mais seul A peut déchiffrer grâce à sa clef privée $K_{prA}$. Le chiffrement à l'aide d'un algorithme à clef public est illustré par la figure I.3.9.

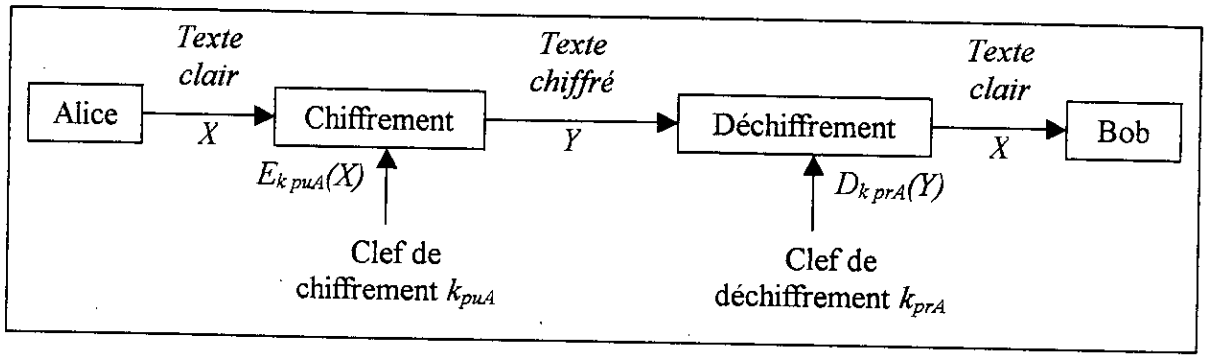



Figure I.3.9 : Chiffrement à l'aide d'un algorithme à clef public

- **Authenticité** : si un message est chiffré par un utilisateur avec la clef $K_{prA}$, alors chacun peut le déchiffrer avec $K_{PuA}$, mais seul A est capable de le chiffrer, car il est le seul à connaître $K_{prA}$. Ce principe est donné dans la figure I.3.10.

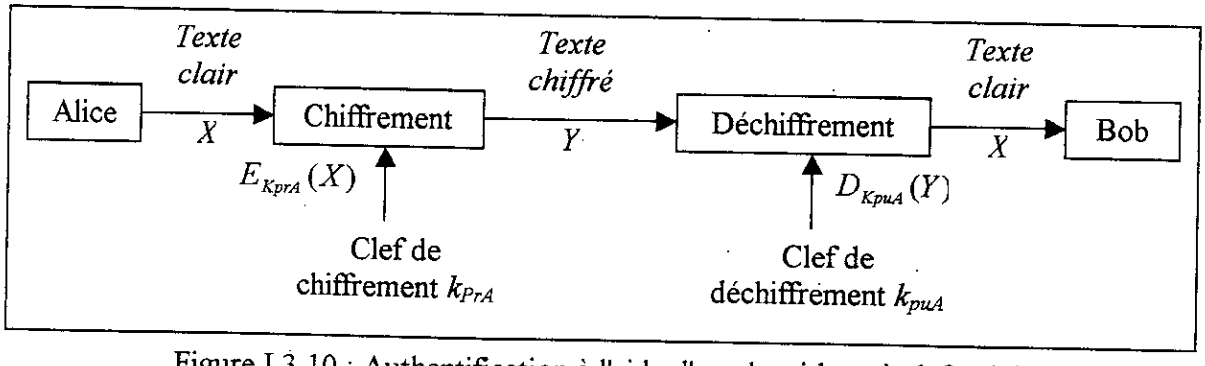



Figure I.3.10 : Authentification à l'aide d'un algorithme à clef public

### b.1) Gestion des clefs dans un cryptosystème à clef publique

Pour illustrer la génération des clefs dans un cryptosystème à clef publique nous prenons comme exemple le système de **chiffrement RSA** [2] qui est considéré comme étant la première réalisation des systèmes à clef publique et fut publiée en 1977 par Rivest, Shamir et Adleman. Le chiffrement RSA est basé sur la difficulté de la factorisation des grands entiers. La description formelle du système est donnée dans la figure I.3.11.

Soit $n=pq$ où $p$ et $q$ sont premiers. Soit $P=C=Z_n$. On définit

$$K=\{(n,p,q,a,b): n=pq, p,q \text{ premiers}, ab\equiv 1 \pmod{\phi(n)}\}.$$

Pour $K=(n,p,q,a,b)$, on définit

$$e_k(x)=x^b \bmod n$$

et

$$d_k(y)=y^a \bmod n$$

$(x,y\in Z_n)$. Les valeurs $n$ et $b$ sont publiques, et les valeurs $p$, $q$, et $a$ sont secrètes.

Figure I.3.11 : Le chiffrement RSA

La génération des clefs dans le système de chiffrement RSA se fait comme suit :

a. Bob engendre deux grands nombres premiers p et q.
b. Bob calcule $n=pq$ et $\phi(n)=(p-1)(q-1)$.
c. Bob choisit un nombre $b$ aléatoire ($1<b<\phi(n)$) tel que $pgcd(b,\phi(n))=1$.
d. Bob calcule $a=b^{-1} \bmod \phi(n)$ en utilisant l'algorithme d'Euclide.
e. Bob publie $n$ et $b$ dans un répertoire

La sécurité de RSA est basée sur l'hypothèse que la fonction $e_K(x)=x^b \bmod n$ est à sens unique [2], donc qu'il est impossible à un opposant de décrypter un texte chiffré.

### b.2) Les certificats numériques sur les clefs publiques

Les certificats numériques sont importants en cryptographie à clé publique, parce qu'ils garantissent qu'un intermédiaire n'espionne pas les échanges. Le principe de fonctionnement de cette procédure est montré dans la figure I.3.12.

Supposons qu'Alice veuille envoyer à Bernard un message authentifié. En utilisant sa clé privée, Alice crée une signature numérique qu'elle joindra au message. Puis, en utilisant la clé publique d'Alice, Bernard vérifiera cette signature. Cependant, comment Bernard saura-t-il que la clé publique appartient effectivement à Alice?

Un imposteur pourrait créer sa propre paire de clés et envoyer sa clé publique à Bernard en prétendant qu'elle appartient à Alice. On évite cette fraude en utilisant une autorité de certification, qui fournira un certificat numérique sur la clé publique reconnu par tous.

1. Alice crée une clé privée (a) ainsi qu'une clé publique (b). Elle envoie la clé publique à une autorité de certification, à qui elle demande un certificat numérique. Pour authentifier Alice, cet organisme vérifie les informations fournies par Alice. Si tout est en ordre, l'organisme de certification envoie un certificat numérique (c) qui authentifie la clé publique d'Alice. Sur ce certificat se trouve la signature numérique du tiers de confiance qui peut être vérifiée par toute personne connaissant la clé publique de cet organisme.

2. La clé publique (d) de l'autorité de certification est fournie à ceux qui en ont besoin, dont Bernard. Elle peut être incluse dans les programmes de navigation sur le réseau Internet et dans d'autres logiciels utilisés pour les communications informatiques sécurisées.

3. Alice signe numériquement le message qu'elle envoie à Bernard. Cette signature (e) est envoyée à Bernard en même temps que le message (f). Alice envoie aussi son certificat numérique, qui contient sa clé publique.

4. Bernard utilise la clé publique correspondant à l'autorité de certification pour vérifier la signature numérique officielle apposée sur le certificat. Il est alors certain que le certificat est authentique et que la clé publique qui l'accompagne appartient à Alice. Il utilise alors cette clé pour déchiffrer la signature numérique d'Alice.

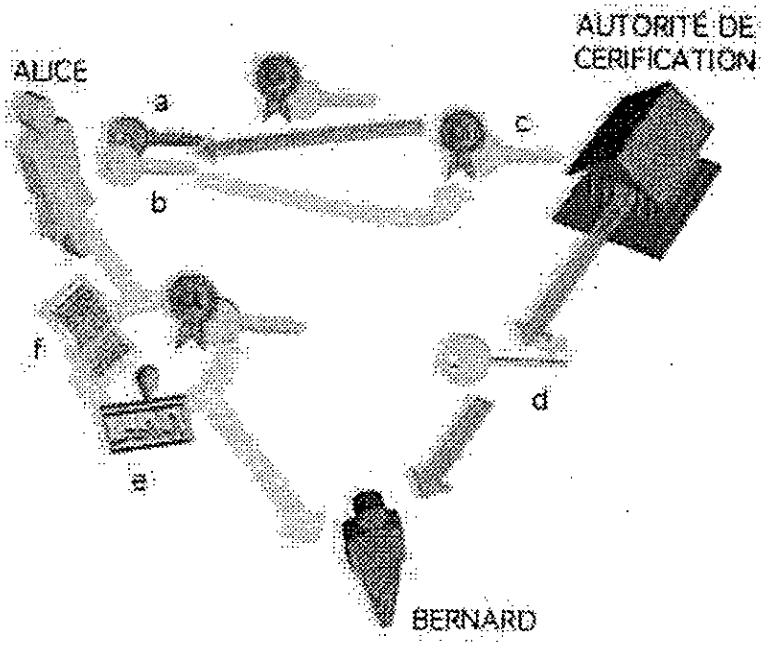



Figure I.3.12 : Les certificats numériques

## I.3.5 Conclusion

A travers ce chapitre, nous avons donc pu discuter des différentes considérations à prendre en compte pour la réalisation d'un cryptosystème le plus solide que possible. Ceci peut être résumé par les différents points suivants :

- Le fait de communiquer à travers un canal non sûr nous oblige à choisir un protocole cryptographique , variant selon le type d'informations à échanger, afin d'exclure des tiers malveillants.

- La sécurité de notre cyptosystème doit résider dans la clef et non pas dans les détails de l'algorithme. Ceci nous amène à poser des conditions sur le choix des clefs et leurs gestions.

- Le choix du mode de chiffrement dépend du type de réalisation (logicielle ou matérielle), du type de données chiffrées et du niveau de sécurité imposé.

- L'utilisation d'un cryptosystème hybride utilisant un algorithme à clef secrète pour le chiffrement de données et un algorithme à clef publique pour l'échange de clefs.

- La prédominance des cryptosystèmes à réalisations matérielles par rapport aux cryptosystèmes à réalisations logicielles pour des raisons de vitesse et de sécurité.

Pour le choix entre un cryptosystème à clef publique et un cryptosystème à clef secrète, un certain nombre de questions doivent être posées. Quel est le meilleur ?

Dans quel cas on utilise l'un par rapport à l'autre ? Nous allons répondre à toutes ces questions à travers les différents points suivants :

1. La force de la cryptographie à clef publique réside dans la distribution de clefs et dans les protocoles. Les algorithmes à clef publique sont idéaux pour chiffrer et distribuer des clefs, permettre l'authentification et effectuer n'importe quelle fonction cryptographique. Les problèmes de l'algorithme à clef publique sont les suivants :

   - Certains algorithmes à clef publique sont 1000 fois plus lents que les algorithmes à clef secrètes et nécessitent des clefs presque 10 fois plus longues [30, 31].

   - L'études [32] montre que la cryptographie à clef secrète est plus sûre que la cryptographie à clef publique.

2. La cryptographie à clef secrète est infiniment plus rapide. Les algorithmes à clef secrète sont idéaux pour chiffrer des fichiers et des canaux de communications. Les problèmes de l'algorithme à clef secrète sont les suivants :
   - Si la clé secrète est compromise (volée, extorquée, piratée, ...) par un opposant, alors ce dernier pourra déchiffrer tous les messages chiffrés avec celle-ci. Oscar, un attaquant, peut même se faire passer pour Alice ou Bob.

   - Les clés doivent être distribuées secrètement : c'est très difficile à l'échelle planétaire (se rencontrer, utiliser un messager sûr, etc...).

   - Si une clé différente est utilisée pour chaque paire différentes d'utilisateurs d'un réseau, le nombre total des clés augmente très rapidement en fonction du nombre total d'utilisateurs.

Ainsi, on peut conclure que pour qu'un cryptosystème fonctionne au mieux, on doit utiliser l'association de la cryptographie à clef secrète et de la cryptographie à clef publique. Ce principe est illustré par la figure I.3.13.

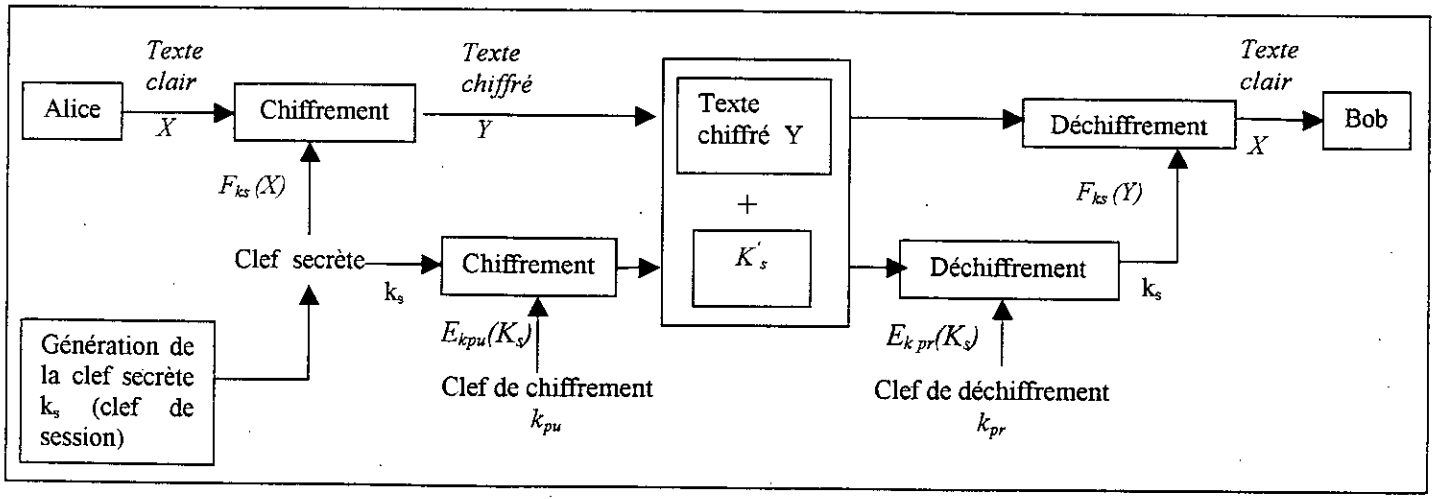



Figure I.3.13 : Solution de chiffrement mixte

où F est une fonction de chiffrement symétrique et E est une fonction de chiffrement asymétrique.
Nous nous intéresserons dans notre cas à la construction de la fonction cryptographique à clef secrète; ce qui constitue un premier travail d'implémentation matérielle d'un cryptosystème mixte. Ceci nous emmène à consacrer le sous-chapitre suivant à l'étude détaillée de l'algorithme DES pour son implémentation dans une structure microprocesseur.

# I.4 - L'Algorithme DES

Le standard de chiffrement de données ou DES a été publié par la National Bureau of standards en 1975. DES est aussi appelé algorithme de chiffrement par bloc, c'est à dire, qu'il chiffre et déchiffre un bloc de données binaire à la fois, à l'opposé des algorithmes de chiffrements en continu qui chiffrent et déchiffrent une chaîne binaire en procédant bit par bit. La figure I.4.1 montre le diagramme de base de cet algorithme.

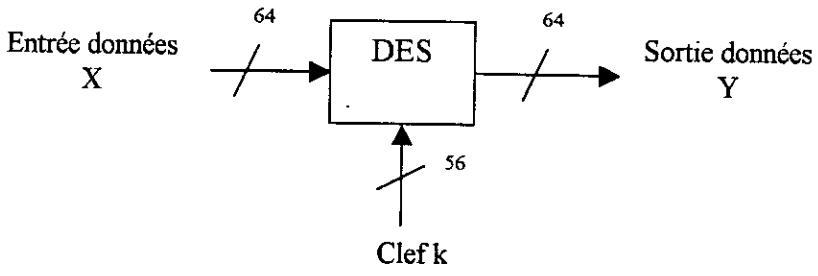



Figure I.4.1 : Vue d'ensemble de l'algorithme de chiffrement DES

DES chiffre des blocs de données de 64 bits (texte clair) en utilisant une clef de 56 bits. Le résultat est un texte chiffré de longueur égale au texte clair.
Durant ce chapitre, on concentre notre explication sur la fonction chiffrement du DES ; la fonction de déchiffrement y est presque identique. Elle sera décrite dans la section 5.3. Notre description mettra surtout en valeur les fonctions internes du DES qui sont importantes pour l'implémentation matérielle.

Voici un exemple simple qui illustre le fonctionnement du DES (Voir figure I.4.2). Alice et Bob partagent la même clef k. Alice chiffre le texte clair X et envoie la version chiffrée Y à travers le réseau à Bob. Bob utilisera la même clef et l'inverse de la fonction DES pour retrouver le texte clair X.

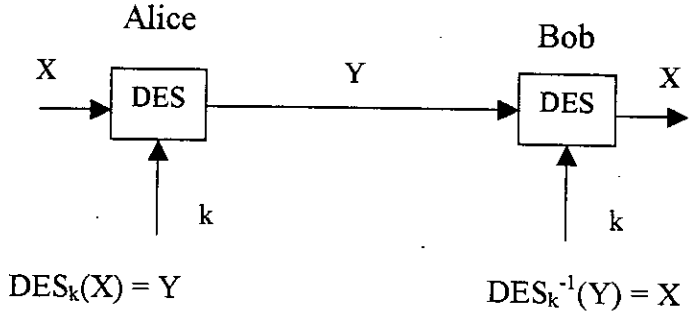



Figure I.4.2 : Principe de fonctionnement de l'algorithme DES

### I.4.1 Les fonctions principales du DES

La figure I.4.3 donne le schéma bloc de l'algorithme DES. Ce dernier se déroule en trois étapes.

1. Etant donné un bloc de texte clair $x$, une chaîne de bits $x_0$ est construite en changeant l'ordre des bits de x suivant une *permutation initiale IP* fixée. On écrit $x_0 = IP(x) = L_0R_0$ où $L_0$ contient les 32 premiers bits de la chaîne $x_0$ et $R_0$ les 32 bits restants.

2. 16 itérations (ou 16 étapes), d'une fonction nommée réseau de Feistel, sont effectuées. On calcule $L_iR_i$, $1 \leq i \leq 16$ suivant la règle :

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Où $\oplus$ représente le ou-exclusif bit-à-bit de deux chaînes. $f$ est une fonction décrite plus loin, et $K_1, K_2, \ldots, K_{16}$ sont des chaînes de 48 bits calculées à partir de $K$ (en fait, chaque $K_i$ est composé de 48 bits donnés de $K$ dans un ordre particulier). On dit $K_1, K_2, \ldots, K_{16}$ sont obtenus par *diversification de la clef.* Une étape de chiffrement est représentée sur la figure I.4.4.

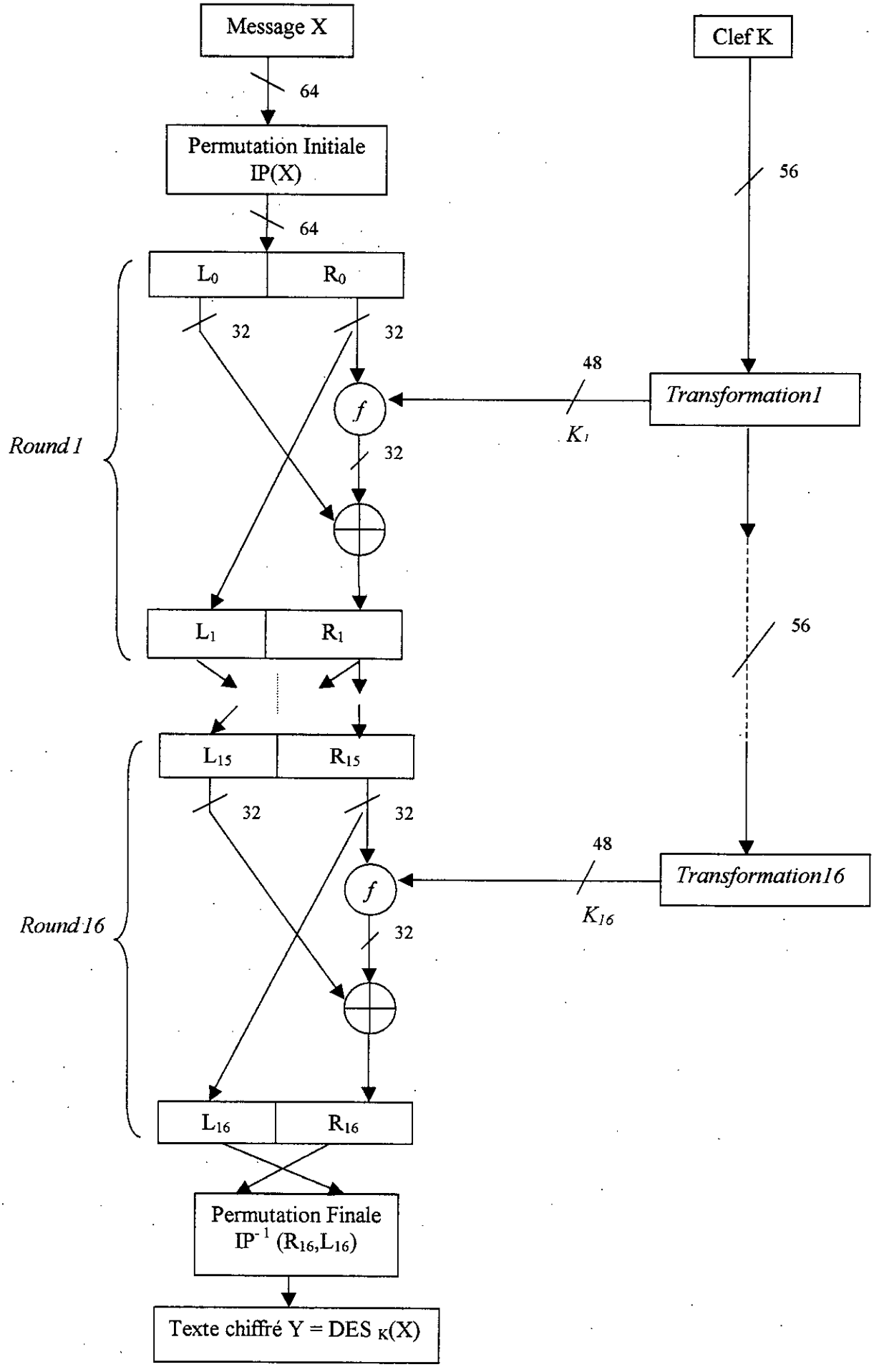



Figure I.4.3 : Schéma bloc de l'algorithme DES

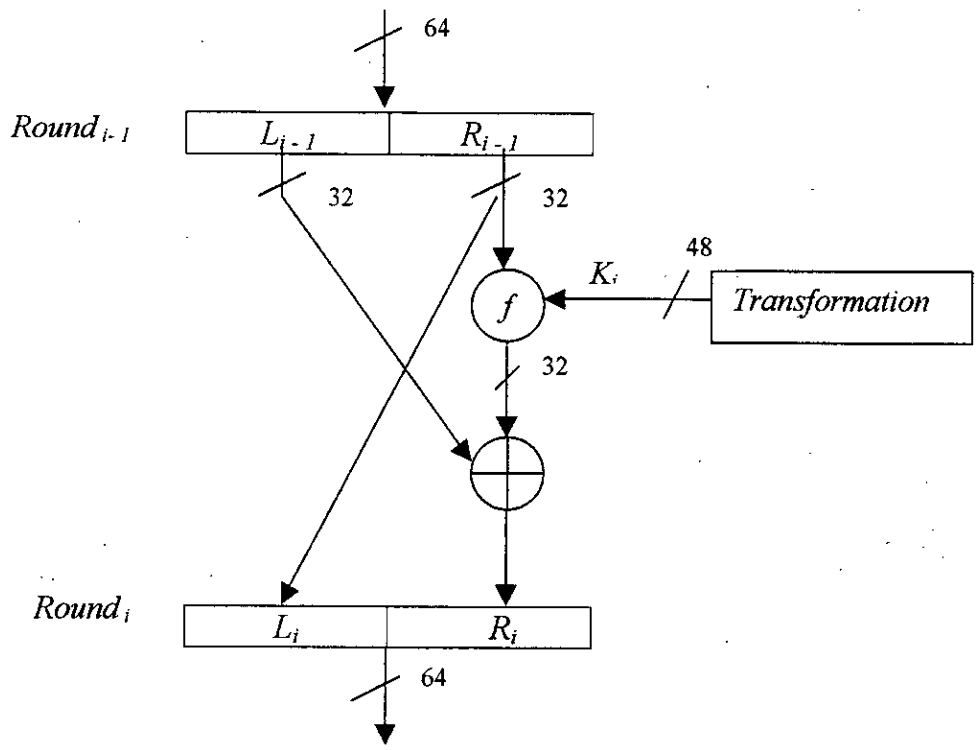



Figure I.4.4 : Une étape de l'algorithme DES (réseau de Feistel du DES)

3. La permutation inverse $IP^{-1}$ est appliquée à $R_{16}L_{16}$ pour obtenir le bloc de texte chiffré $y = IP^{-1}(R_{16}L_{16})$. On note ici l'inversement de $L_{16}$ et $R_{16}$.

Le premier argument $A$ de la fonction $f$ est une chaîne de 32 bits tandis que le second $J$ est de longueur 48, et le résultat de $f$ est sur 32 bits. Le calcul se décompose en plusieurs étapes.

1. Le premier argument $A$ est rallongé en une chaîne de 48 bits suivant une *fonction d'expansion E*. $E(A)$ est composé de tous les bits de $A$ dans un certain ordre ; 16 d'entre eux apparaissent deux fois.

2. $E(A) \oplus J$ est calculé, et le résultat $B$ est découpé en 8 sous-chaînes de 6 bits $B = B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8$.

3. L'étape suivante utilise huit *tables–S* $S_1, \ldots, S_8$. chaque $S_i$ est un tableau 4 x 16 d'entiers compris entre 0 et 15. Etant donné une sous-chaîne de six bits $B_j = b_1 b_2 b_3 b_4 b_5 b_6$, on calcule une chaîne de quatre bits $S_j(B_j)$ ainsi.
Les deux bits $b_1 b_6$ forment la représentation binaire de l'indice d'une ligne $r$ de $S_j$ $(0 \leq r \leq 3)$ et les quatre bits $b_2 b_3 b_4 b_5$ composent la représentation binaire de l'indice d'une colonne $c$ de la table $(0 \leq c \leq 15)$. $S_j(B_j)$ est alors la représentation binaire sur quatre bits du coefficient $S_j(r, c)$. On peut ainsi voir chaque $S_j$ comme une fonction qui admet en entrée une chaîne de six bits (ou une chaîne

de deux et une chaîne de quatre) et produit une chaîne de quatre bits. De cette manière, on calcule $C_j = S_j(B_j)$, $1 \leq j \leq 8$.

4. La chaîne $C = C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8$ de longueur 32 est réordonnée suivant une permutation fixée P. Le résultat P($C$) définit $f(A, J)$.

La fonction $f$ est illustrée sur la figure I.4.5. En fait, c'est une substitution (définie par les tables–S) suivie d'une permutation P.

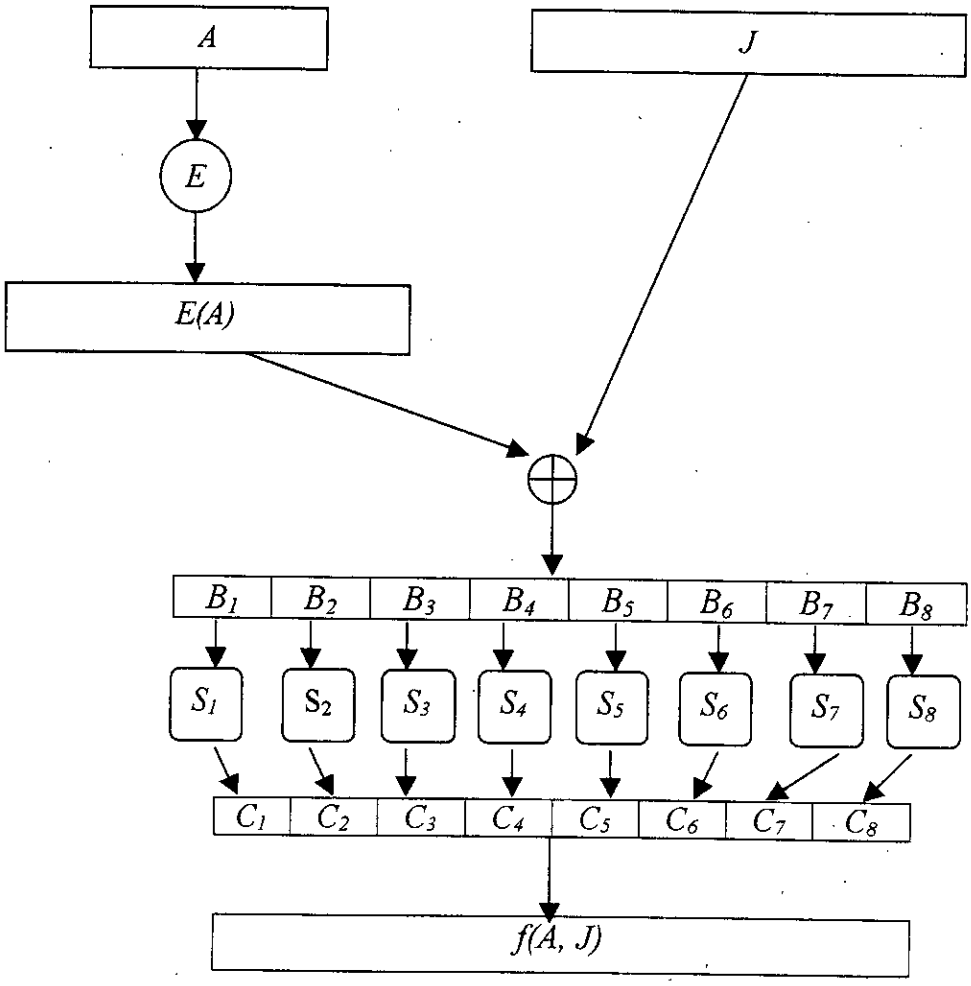



Figure I.4.5 : La fonction $f$ de l'algorithme DES

## I.4.2 Diversification de clef dans l'algorithme DES

Il ne reste qu'à décrire le procédé de diversification de la clef $K$. En fait, $K$ est une chaîne de 64 bits, dont 56 définissent la clef, et 8 sont des bits de parité (ou bits de détection d'erreur). Les bits en position 8,16, ...,64 sont tels que chaque octet contient un nombre impair de 1. Ainsi, une seule erreur peut être détectée dans chaque groupe de 8 bits. Les bits de parité sont ignorés dans le procédé de diversification.

1. Etant donnés les 64 bits de $K$, on enlève les bits de parité et l'on ordonne les autres suivant une permutation PC-1. On note PC-1($K$) = $C_0D_0$ où $C_0$ est composée des 28 premiers bits de PC-1($K$) et $D_0$ des 28 restants.

2. Pour $i$ compris entre 1 et 16, on calcule

$$C_i = LS_i(C_{i-1})$$
$$D_i = LS_i(D_{i-1})$$

Et $K_i$ = PC-2($C_iD_i$). $LS_i$ est une rotation circulaire (vers la gauche) d'une ou deux positions, suivant la valeur de i : on décale d'une position si $i$ = 1, 2, 9 ou 16, et on décale de deux positions sinon. PC-2 est une autre permutation des bits de chaîne.

Le calcul de la diversification de clef est illustré sur la figure I.4.6.

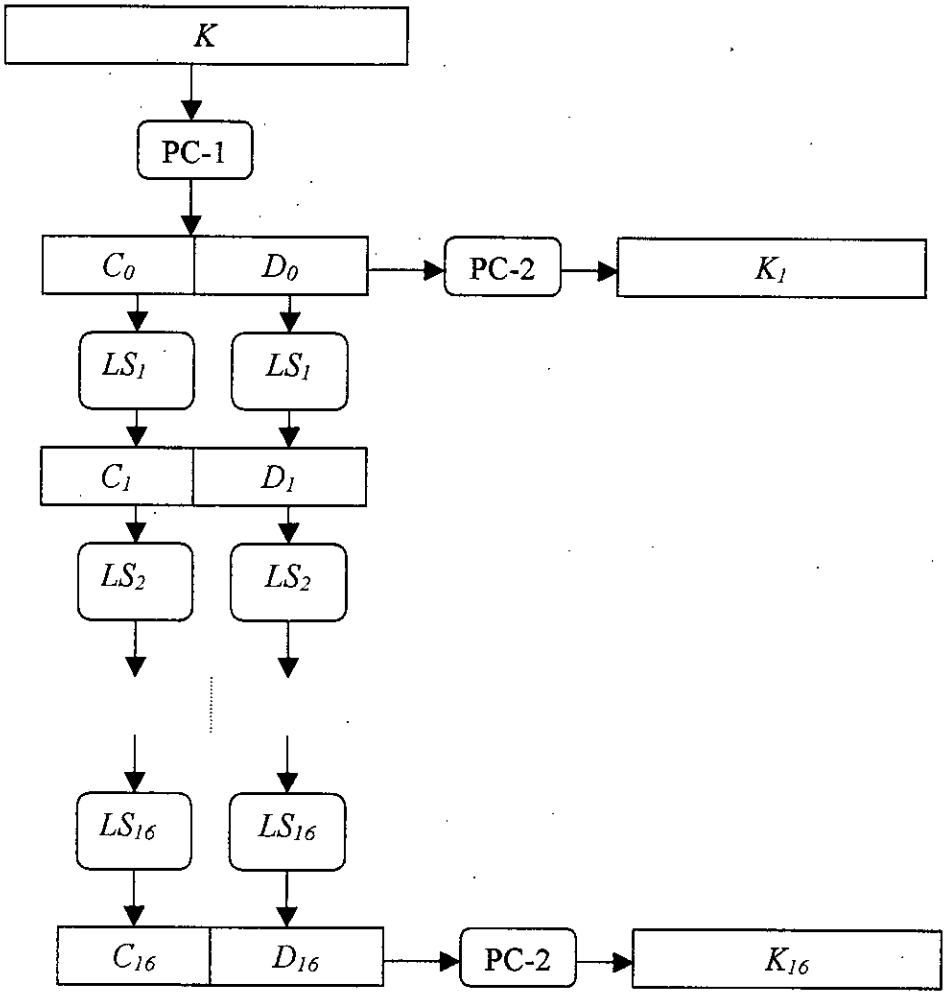



Figure I.4.6 : Diversification de clef dans l'algorithme DES

## I.4.3 Déchiffrement

Le déchiffrement DES s'effectue en utilisant le même algorithme que dans le chiffrement en commençant avec le texte chiffré $y$ et en utilisant les clefs $K_{16}, \ldots, K_1$ dans l'ordre inverse. Le résultat est le texte clair $x$. Dans le but de créer les sous-clefs dans l'ordre inverse, $C_i$ et $D_i$ doivent subir une rotation circulaire vers la droite, à l'opposé du chiffrement où ils subissent une rotation circulaire vers la gauche, suivant la valeur de $i$.

Le tableau suivant montre combien de positions $C_i$ et $D_i$ doivent subir de rotation.

| **Itération** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Chiffrement** | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| **Déchiffrement** | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Tableau I.4.1 : Nombre de rotation des sous-clefs.

## I.4.4 Modes de fonctionnement de l'algorithme DES

Quatre modes d'utilisation de DES furent proposées [2] : *electronic codebook mode* (ECB), *cipher feedback mode* (CFB), *cipher block chaining mode* (CBC) et *output feedback mode* (OFB).

### a) Mode Electronic Codebook (ECB)

C'est l'approche la plus simple pour l'utilisation du chiffrement par blocs. Le texte clair est divisé en blocs $X_i$ de 64 bits et chaque bloc est chiffré séparément (voir figure I.4.7). Le chiffrement d'un même bloc de texte clair donne toujours le même bloc de texte chiffré : $Y_i = e_k(X_i)$. Le problème majeur avec l'utilisation de ce simple mode, est le fait qu'il soit sujet aux attaques par substitution [1](attaques par blocs rejoués).

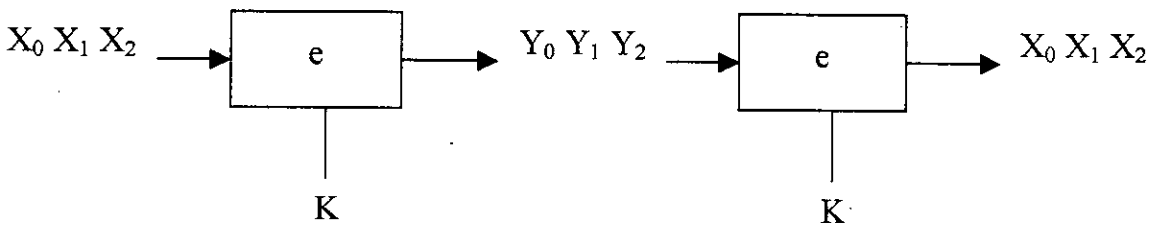



Figure I.4.7 : Electronic Codebook Mode

### b) Mode Cipher Block Chaining (CBC)

Emploi un vecteur d'initialisation IV et une boucle de rétroaction. Chaque bloc chiffré dépend non seulement du bloc de texte en clair qui l'a engendré mais aussi de tous les blocs de texte en clair qui précèdent celui-ci (voir figure I.4.8). Le premier bloc du texte en clair est combiné par un *ou exclusif* avec le vecteur d'initialisation avant d'être chiffré. Chaque bloc du texte clair est combiné par un *ou exclusif* avec le bloc chiffré précédant avant qu'il ne soit chiffré : $Y_0 = e_k(X_0 \oplus IV)$ et $Y_i = e_k(X_i \oplus Y_{i-1})$ pour $i \geq 1$.

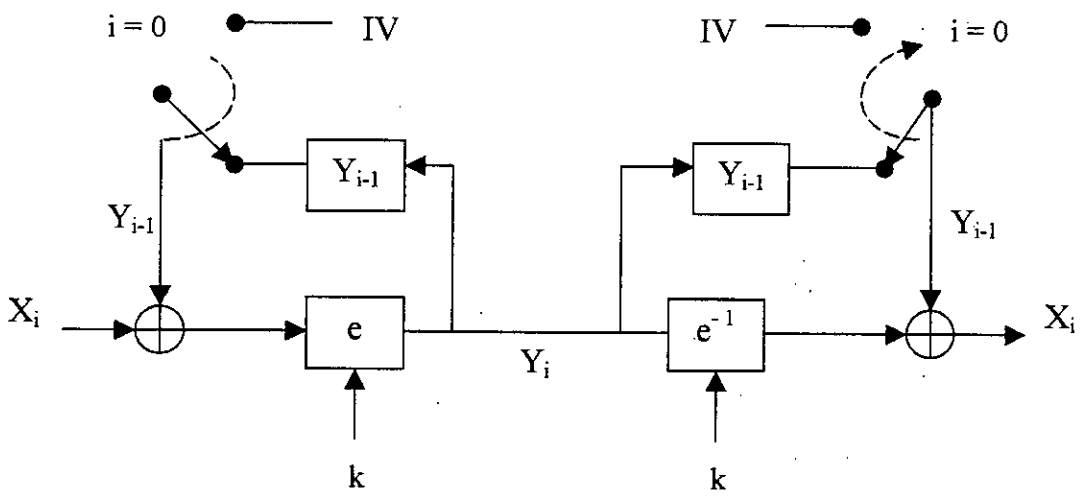



Figure I.4.8 : Mode Cipher Block Chaining

### c) Mode Cipher Feedback (CFB)

Très souvent utilisé pour chiffrer des messages de taille inférieure à 64 bits. La figure I.4.9 montre le schéma du CFD. Un registre à décalage est préchargé avec le vecteur d'initialisation IV à l'étape $i = 0$. La sortie parallèle du registre à décalage de 64 bits est chiffrée : $\tilde{z}_0 = e_k(IV)$.

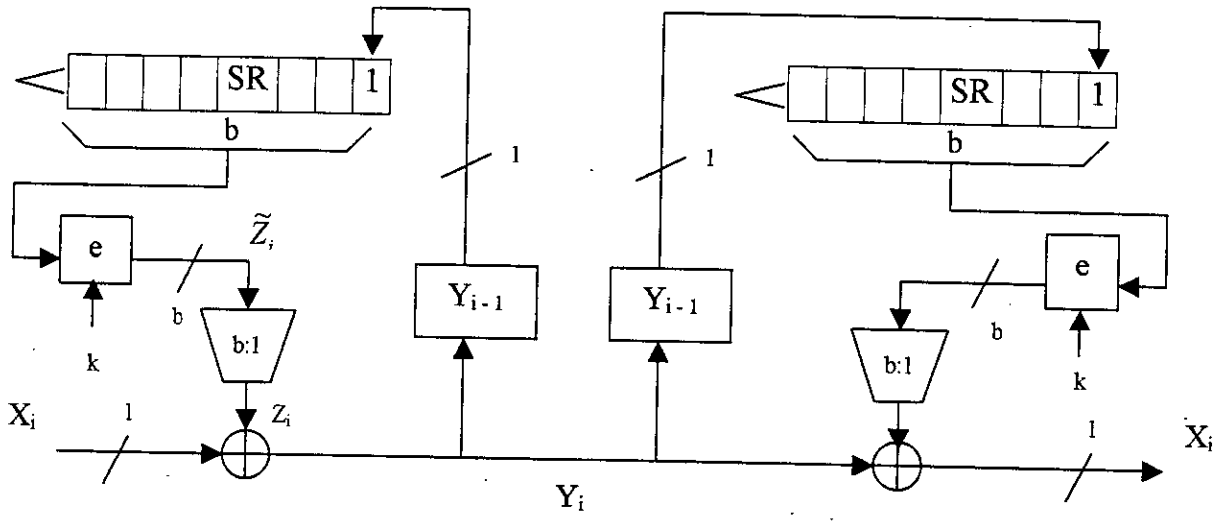



Figure I.4.9 : Mode Cipher Feedback

On prend les $l$ bits les plus à gauche $\tilde{z}_i \rightarrow z_i$ qui sont combinés par un *ou exclusif* avec les $l$ bits du texte clair pour devenir les $l$ premiers bits du texte chiffré : $Y_i = X_i \oplus z_i$. Le registre à décalage subit ensuite un décalé de $l$ bits et le $Y_i$ sera chargé dans la position la plus à droite. Le chiffrement du nouveau contenu du registre à décalage produira le nouveau $\tilde{z}_{i+1}$.

### d) Mode Output Feedback

Similaire au mode CFB, sauf qu'une partie du bloc de sortie précédent est mise dans les positions les plus à droite de la queue et non pas celle du bloc chiffré.

## I.4.5 Amélioration du DES

Pour augmenter le niveau de sécurité de DES, on l'utilise trois fois de suite pour chiffrer le même texte clair. On parlera alors du DES *triple*.

Généralement, il existe deux types de DES *triple* : *chiffrement-déchiffrement-chiffrement* et *chiffrement-chiffrement-chiffrement*.

Pour le type *chiffrement-déchiffrement-chiffrement* souvent deux clefs sont utilisées. Le texte clair $X$ subit un chiffrement avec la première clef $e_{k1}(X)$, suivi d'un déchiffrement avec la seconde clef $e_{k2}^{-1}(e_{k1}(X))$ et en dernier chiffré avec la première clef : $Y = e_{k1}(e_{k2}^{-1}(e_{k1}(X)))$.

La figure I.4.10 montre le type chiffrement-chiffrement-chiffrement. Le texte clair $X$ subit un chiffrement trois fois de suite avec trois clefs différentes : $Y = e_{k3}(e_{k2}(e_{k1}(X)))$.

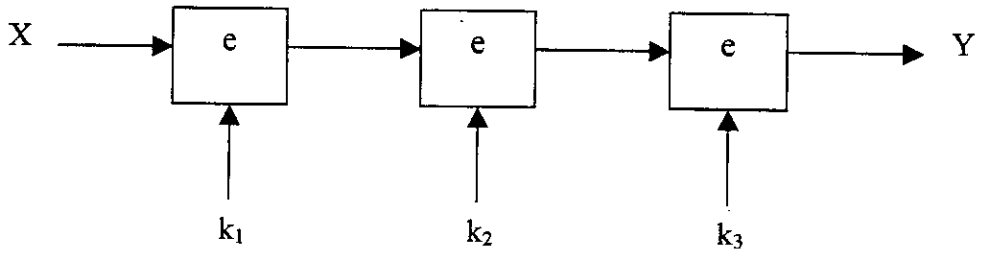



Figure I.4.10 : DES *triple*

## I.4.6 Conclusion

La principale conclusion à tirer de l'étude théorique de l'algorithme DES est que celui-ci possède une structure itérative à partir de laquelle nous pouvons tirer une structure de base. Cette structure nous aide à déduire des options architecturales qui auront pour but d'accélérer l'exécution de l'algorithme. Tout ce travail d'optimisation sera exposé dans le chapitre suivant.

# Chapitre II

# Implémentation de l'algorithme DES sur FPGA

## II.1 Introduction

La démarche qui sera poursuivie pour l'implémentation de l'algorithme DES sur un composant FPGA doit être conforme aux modèles de conception exprimés au travers du diagramme en Y de Gajski. Par conséquent, cette démarche passera par les trois domaines d'expression, à savoir, le domaine comportemental, le domaine structurel et le domaine physique.

1. **Modèle comportemental de l'algorithme DES** : Suite à l'étude faite sur l'algorithme DES où il en ait ressorti que ce dernier possède une structure itérative. Nous amène dans un premier temps à procéder à sa structuration afin de dégager une structure de base à partir de laquelle découlera des options architecturales. Ces dernières viseront à augmenter les performances notamment la réduction du temps de calcul de l'algorithme DES.

2. **Modèle structurel associé à l'algorithme DES** : La conception du circuit implique la mise en place de deux entités qui sont : le chemin de données et le séquenceur. Afin de pouvoir réaliser des optimisations et un gain en délais et en ressources logiques à consommer du circuit FPGA cible, le chemin de données sera divisé en blocs fonctionnels. Cette approche de conception en Down-Top (du Bas vers le Haut) aura un deuxième avantage. Celui-ci facilitera énormément la conception des différentes options architecturales de l'algorithme DES. Ainsi, pour la réalisation d'une nouvelle conception il suffira d'ajouter ou de soustraire un des blocs fonctionnels.

3. **Implémentation du modèle structurel associé à l'algorithme DES** : Avant de procéder à l'implémentation de l'algorithme DES et de ses options architecturales, nous commençons par le choix du composant FPGA cible en tenant compte des contraintes liées à la structure et la nature de certains blocs fonctionnels. L'algorithme DES sera implémenté en plusieurs versions dont chacune correspondra à une option architecturale donnée, et ce dans le but de pouvoir comparer les performances d'une option architecturale par rapport à une autre. Pour chaque version à implémenter nous devons prévoir la logique de commande correspondante.

# II.2 - Modèles comportementaux de l'algorithme DES

## II.2.1 Introduction

La première étape à aborder afin d'obtenir une implémentation la plus performante de l'algorithme DES est la structuration de l'algorithme.

## II.2.2 Structuration de l'algorithme DES

Tel que déjà décrit dans le chapitre 2, l'algorithme DES possède une structure itérative. La figure I.4.4 montre le passage des données 16 fois à travers la fonction principale dite réseau Feistel. Pour chaque itération, on fait associer à cette dernière une sous–clef différente qui provient de la partie diversification de clef.

L'organigramme de la figure II.2.1 illustre le fonctionnement de l'algorithme DES. La fonction principale est représentée par l'*itération i*. Dés l'introduction du texte clair X, le compteur d'itération i est mis à 1. Après chaque itération, *i* est testé. S'il est plus petit que 16 il sera incrémenté; la sortie courante est bouclée pour qu'elle soit réintroduite dans le réseau Feistel. La prochaine itération commence. Après 16 itérations le calcul du texte chiffré Y est prêt en sortie.

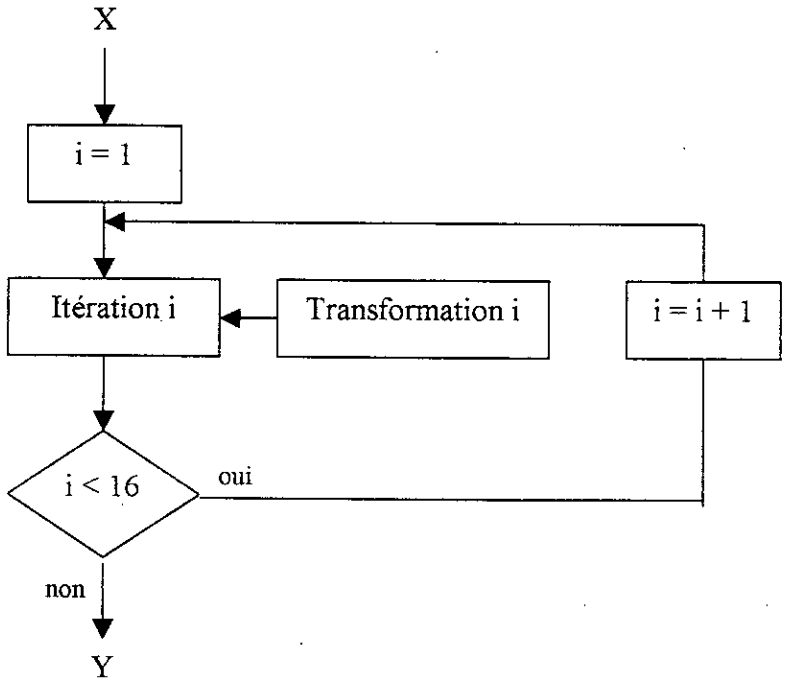



Figure II.2.1 : Organigramme de l'algorithme DES

A partir de cet organigramme, nous pouvons déduire le schéma–bloc du DES (figure II.2.2) qui est très proche de notre implémentation matérielle. Cette structure de base va nous permettre de faire des améliorations afin d'accélérer l'exécution du DES.

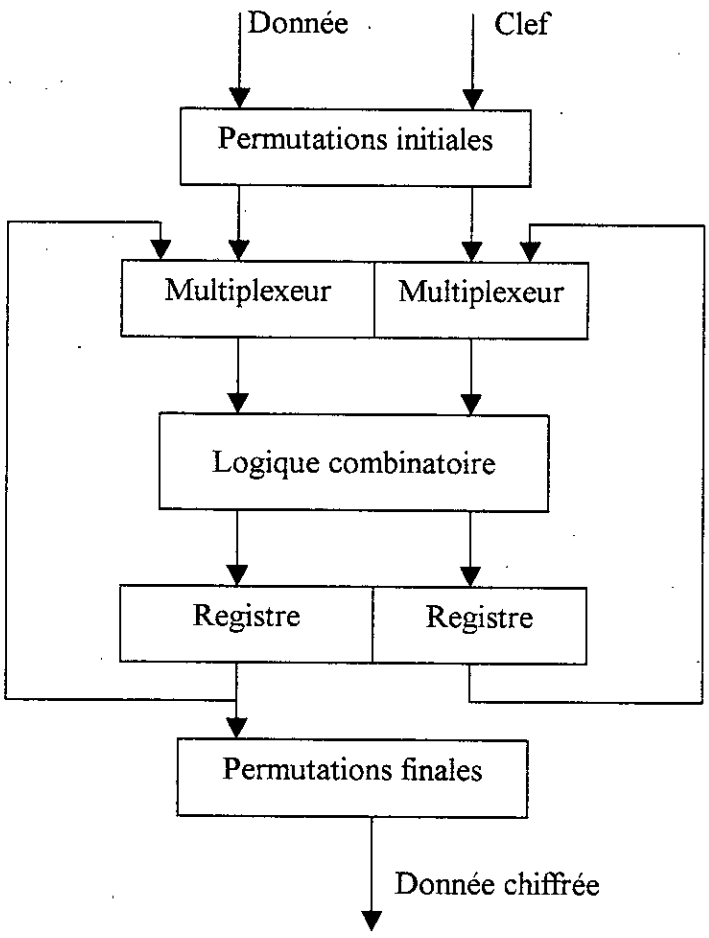



Figure II.2.2 : Schéma–bloc de l'algorithme DES

La donnée et la clef présentent en entrée subissent une permutation initiale. Ensuite la donnée passe 16 fois à travers le réseau Feistel ; 16 clefs sont générées simultanément.

La fonction principale et la génération des sous–clefs sont représentées dans le schéma–bloc par l'unité logique combinatoire (ULC). Afin de pouvoir boucler la sortie sur l'entrée de l'unité combinatoire, on a besoin de registres et de multiplexeurs. Les multiplexeurs servent à sélectionner, à l'entrée de l'unité logique combinatoire, les données de la précédente itération par rapport aux nouvelles données présentent en entrée. Le même principe est appliqué pour les clefs. Les registres stockent le résultat de chaque itération et le font passer aux multiplexeurs. A la fin de chaque itération, la sortie du registre de données subit une permutation finale et les données seront présentent en sortie. Donc une des tâches de la logique de commande est de signaler à une entité extérieure si les données sont valides ou non.

## II.2.3 Techniques d'accélérations du DES

Après avoir déduit la structure de base du DES et étudié son fonctionnement, on remarque que des optimisations peuvent être faites dans le sens de la réduction du temps de calcul. Par conséquent, certaines techniques d'accélération du temps de traitement ont été apportées à cette structure de base.

### a) Structure en boucle expansée

La première technique utilisée pour accélérer l'implémentation matérielle de l'algorithme DES est la structure en boucle expansée. Elle consiste en la concaténation de deux unités logiques combinatoires, afin de réduire à moitié le nombre d'itérations. Par conséquent, en un seul cycle d'horloge, on calculera deux rounds du DES. La figure II.2.3 montre le schéma–bloc de cette structure. Elle diffère du schéma–bloc de la figure II.2.2 seulement par l'ajout de la seconde unité logique combinatoire. Les permutations initiales et finales ainsi que les multiplexeurs et les registres restent identiques.

Où réside l'amélioration de la vitesse ?
Dans la version du DES à structure standard, une itération a le modèle de temps suivant : $T_{mux} + T_{cl} + T_{reg}$ où $T_{mux}$ représente le temps nécessaire à un signal pour traverser un multiplexeur, $T_{cl}$ le retard introduit par la logique combinatoire et $T_{reg}$ le retard introduit par le registre. Pour l'ensemble des 16 itérations on déduit : $16 * T_{mux} + 16 * T_{cl} + 16 * T_{reg}$.

Le modèle de temps pour la version de la structure en boucle expansée possède la forme suivante : $T_{mux} + 2 * T_{cl} + T_{reg}$.
Cette dernière doit être exécutée 8 fois, d'où un délais global de la forme : $8 * T_{mux} + 16 * T_{cl} + 8 * T_{reg}$.

Le même principe peut être appliqué pour la version du DES avec une structure en boucle à quatre (04) expansions. La liste suivante montre l'équation temporelle pour chaque cas :

DES en structure de base : $16 * T_{mux} + 16 * T_{cl} + 16 * T_{reg}$
Structure en boucle à deux (02) expansions : $8 * T_{mux} + 16 * T_{cl} + 8 * T_{reg}$
Structure en boucle à quatre (04) expansions : $4 * T_{mux} + 16 * T_{cl} + 4 * T_{reg}$

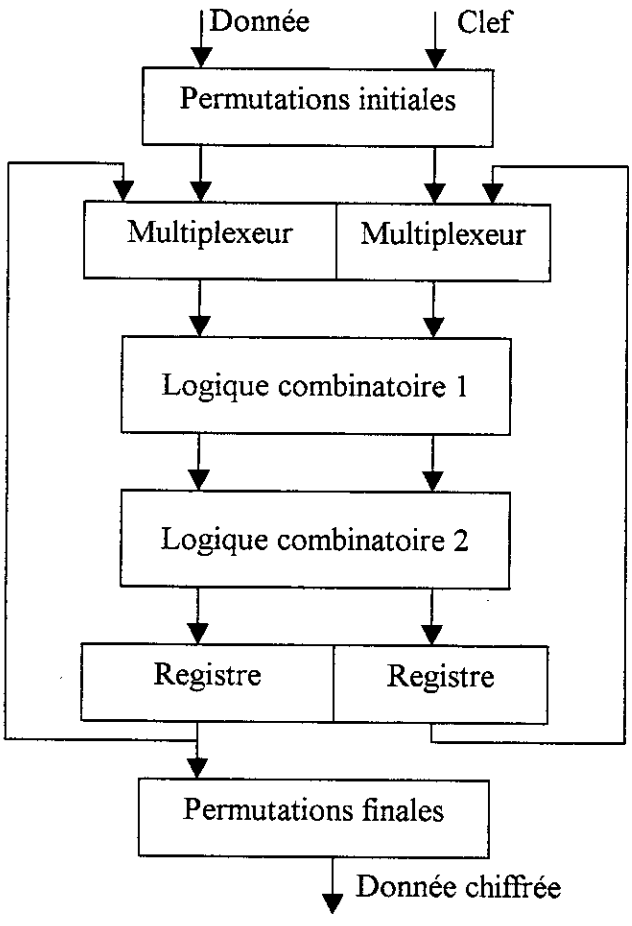



Figure II.2.3 : Schéma–bloc de l'algorithme DES avec une structure en boucle expansée

Bien sûr, par l'utilisation de cette technique, nous ne pouvons pas réduire le délai introduit par l'unité logique combinatoire, mais nous sommes parvenus à réduire de moitié le temps de passage à travers les multiplexeurs et les registres. Il existe cependant d'autres moyens pour augmenter la vitesse de traitement en faisant recours à des outils de synthèses très performants optimisant potentiellement les unités logiques combinatoires d'où une diminution accrue du temps de traitement global des différentes structures proposées.

**b) Structure en Pipeline**

Nous allons étudier maintenant l'architecture principale pour l'accélération du DES. Avec cette structure en pipeline, nous allons essayer d'améliorer la vitesse par un chemin différent. Au lieu de traiter un bloc de données en un temps t donné, la conception en pipeline peut traiter deux ou plusieurs blocs à la fois. Une structure avec deux pipelines est illustrée par la figure II.2.4. Le schéma–bloc de la figure II.2.4 est très similaire avec celui de la structure en boucle à deux (02) expansions (figure II.2.3). La seule différence est l'ajout de registres entre les unités logiques combinatoires.

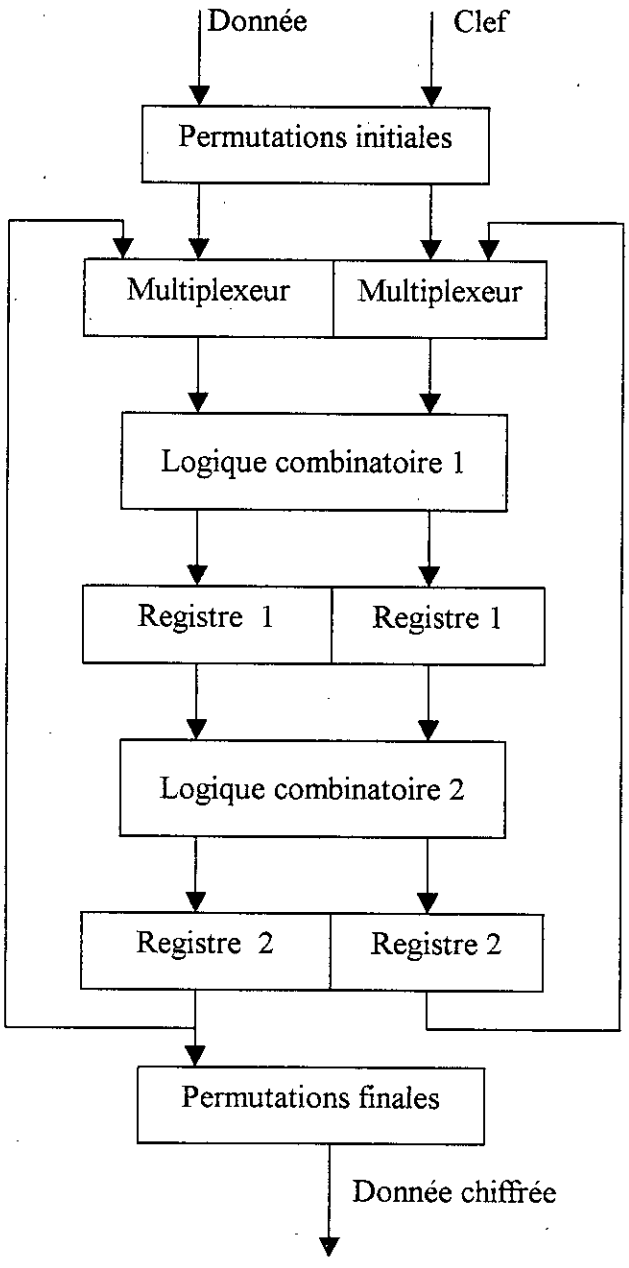



Figure II.2.4 : Schéma–bloc de l'algorithme DES avec une structure en pipeline

Le premier bloc de donnée $x_1$ et la clef associée $k_1$ sont chargées et passent à travers les permutations initiales et le multiplexeur. La première unité logique combinatoire calcul $x_{1,1}$ et $k_{1,1}$. Ils sont stockées par la suite dans le premier bloc de registres. Au deuxième cycle d'horloge $x_{1,1}$ et $k_{1,1}$ quittent les premiers registres et la deuxième unité logique combinatoire calcul $x_{1,2}$ et $k_{1,2}$ et placées à leur tour dans le second bloc de registres. Et en même temps le second bloc de donnée $x_2$ et la clef $k_2$ sont chargées et passent à travers les permutations initiales et le multiplexeur, et la première unité combinatoire calcul $x_{2,1}$ et $k_{2,1}$ qui accèdent ensuite au premier bloc de registre.

Maintenant que le pipeline est rempli, à chaque cycle d'horloge on calcul deux paires données–clefs. La donnée qui entre la première dans le pipeline, sera la première à le quitter. Et en même temps la prochaine paire donnée–clef pourra être chargée.

Les avantages de cette conception peuvent être résumés dans les points suivants :

- Deux ou plusieurs paires donnée–clef peuvent être calculées en même temps.
- Du point de vue de l'implémentation et si on devait comparer l'implémentation de cette structure en pipeline avec une implémentation comprenant deux structures standards du DES sans pipeline, on aurait tiré les avantages suivants :

    - Une implémentation en pipeline utiliserait moins de ressources, par le fait que certains blocs fonctionnels, tels que, les permutations initiales, multiplexeurs et permutations finales ne sont utilisés qu'une seule fois.

    - L'existence d'une seule logique de contrôle légèrement plus complexe.

    - Il est recommandé de concevoir des structures en pipeline qui dépassent les deux étages, par exemple, avec quatre étages afin d'avoir une vitesse de traitement plus importante.

**c) Combinaison des structures Pipelines et en boucle expansée**

Il est possible de combiner les deux techniques d'accélérations, déjà décrites, où chaque pipeline contient deux unités combinatoires concaténées. Le résultat est le schéma–bloc montré en figure II.2.5 qui paraît similaire à la figure II.2.4 sauf que chaque unité logique combinatoire est dupliquée. Durant un cycle d'horloge, deux itérations de deux paires données–clefs sont calculées : $x_{1,4}$ et $k_{1,4}$ sont calculées à partir de $x_{1,2}$ et $k_{1,2}$. $x_{2,2}$ et $k_{2,2}$ sont calculées à partir de $x_2$ et $k_2$.

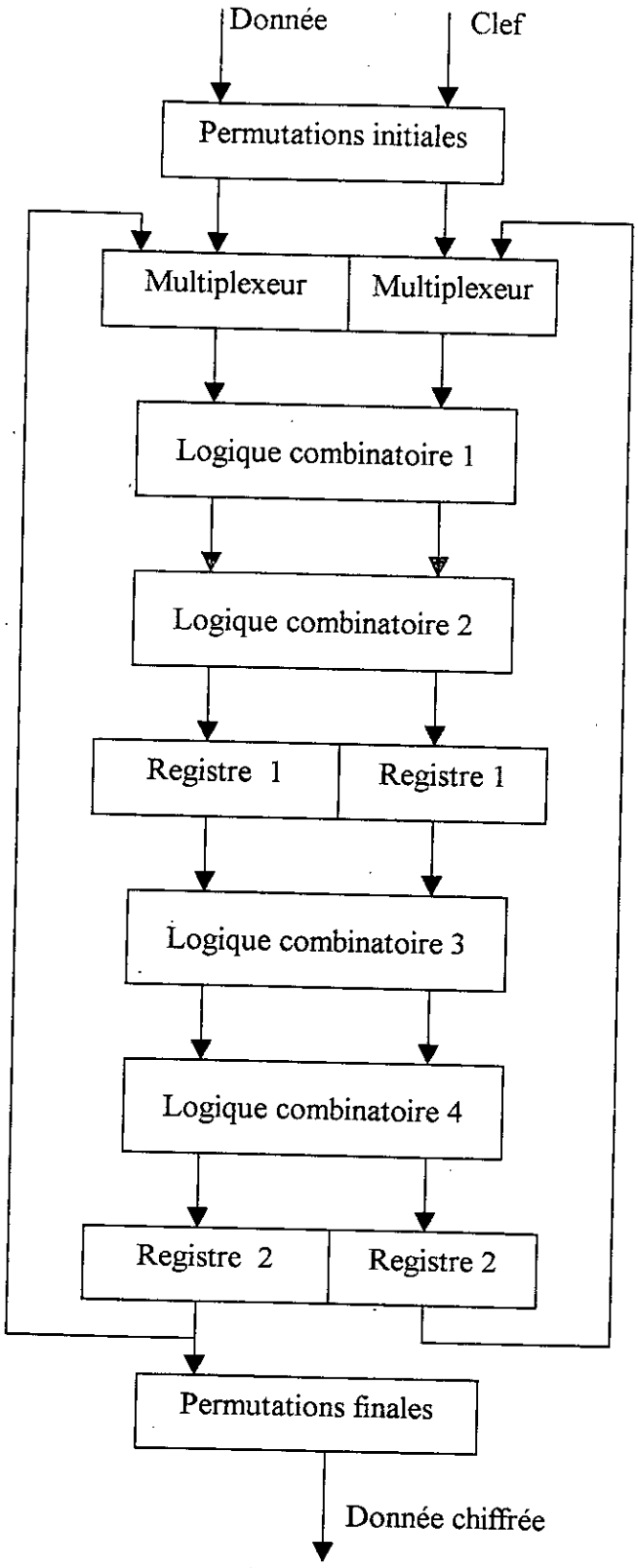



Figure II.2.5 : Schéma–bloc de l'algorithme DES avec une structure en boucle expansée et en pipeline

## II.2.4 Conclusion

Tel que déjà décrit précédemment, certains modes de fonctionnement du DES exigent que la sortie soit utilisée pour le calcul de la prochaine entrée, par exemple, le mode CFB. Si un tel mode devait être utilisé, une conception en pipeline ne fonctionnera pas, parce que une telle conception traite deux paires données–clefs en même temps. Une conception en boucle expansée doit très bien fonctionner et en plus c'est la seule méthode qui peut être appliquée pour augmenter la vitesse de traitement pour les modes ayant un fonctionnement en rétroaction. Pour les applications qui ne sont pas soumises à ce type de contraintes, le mode ECB par exemple, la version pipeline peut être utilisée surtout que la conception en pipeline aboutit à des accélérations de traitement, supérieures à celles obtenues par les conceptions en boucle expansée.

Le tableau II.2.1 montre les versions d'architectures qu'on a décidé d'implémenter.

| Nom | Description |
|---|---|
| DES_VER1 | DES standard (16 itérations) |
| DES_VER2.1 | DES avec une structure en boucle à deux (02) expansions (8 itérations) |
| DES_VER2.2 | DES avec une structure en boucle à quatre (04) expansions (4 itérations) |
| DES_VER3.1 | DES avec une structure à 2 pipelines (16 itérations) |
| DES_VER3.2 | DES avec une structure à 4 pipelines (16 itérations) |
| DES_VER4 | DES avec une structure combinée, pipeline et en boucle expansée |

Tableau II.2.1 : Architectures possibles d'implémentation du DES

# II.3 - Modèle structurel associé à l'algorithme DES

## II.3.1 Introduction

Après avoir analysé l'architecture de l'algorithme DES, l'étape suivante serait de procéder à la conception du circuit par la mise en œuvre de deux entités : le chemin de données et le séquenceur. Le chemin de données quant à lui sera divisé en blocs élémentaires appelés blocs fonctionnels et ce afin d'analyser leur implémentation et de déterminer les optimisations à effectuer lors de la conception finale.

## II.3.2 Le chemin de données

Dans le chapitre précédant, nous avons monté que la structure de l'algorithme DES était composée des différentes parties : la permutation initiale, la permutation finale, les registres et les multiplexeurs couplés à l'unité logique combinatoire qui est considérée comme étant la partie essentielle de cette structure. L'unité logique combinatoire quant à elle, est constituée du réseau de Feistel et de la fonction de diversification de la clef. Toutes ces parties sus citées constituent en réalité le chemin de donnée de notre conception.

Le réseau de Feistel, tel que montré en figure II.3.1, comprend un XOR de 32 bits et la fonction *f*. La fonction *f* à son tour est composée d'une fonction d'expansion, d'un XOR de 48 bits, de huit (08) tables–S et d'un module de permutation.
La diversification de la clef nécessite des registres à décalages et des modules de permutation. Les registres à décalages auront à réaliser une rotation circulaire sur une ou deux positions en fonction de la valeur de l'itération et change de sens de rotation suivant qu'on est en mode chiffrement ou en mode déchiffrement.

Afin de réaliser toutes les opérations appartenant au chemin de données, nous devons concevoir les blocs fonctionnels suivants (voir figure II.3.1):

- Permutation et expansion
- Registres
- Multiplexeurs
- Fonctions logiques standards (XOR)
- Tables–S
- Registres à décalage

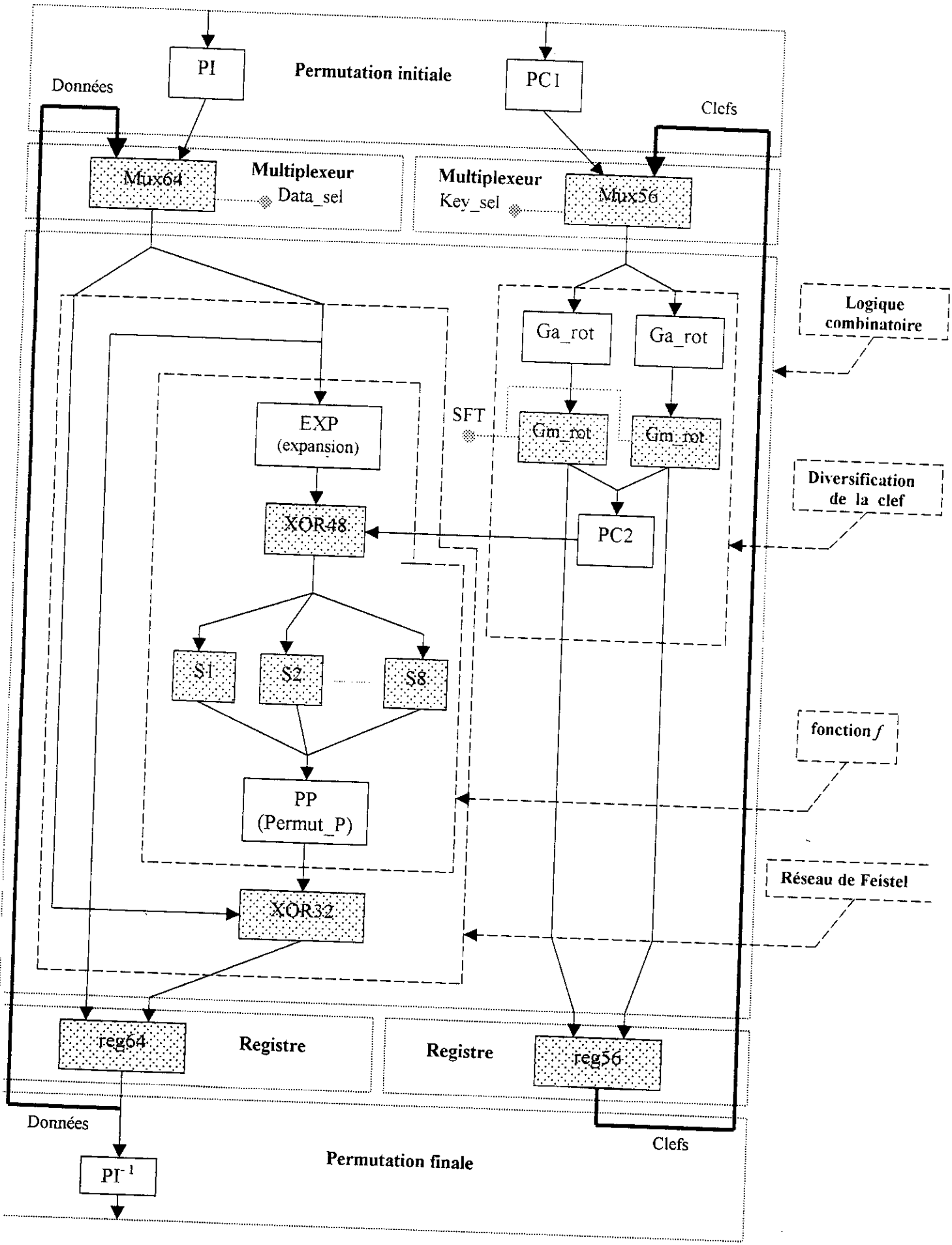



Figure II.3.1 : Schéma bloc détaillé de DES

### II.3.2.1 Signaux de commande du chemin de données

La figure II.3.1 montre les signaux de commande du chemin de données pour la partie commande. Nous trouvons les signaux de commande :

- des multiplexeurs (data_sel et key_sel), pour la sélection entre une nouvelle paire (donnée, clef) et une paire (donnée, clef) bouclée.

- des registres à décalage (SFT)

### II.3.2.2 Ressources logiques des blocs fonctionnels

Chaque bloc fonctionnel cité dans le paragraphe II.3.2 va être analysé et les méthodes utilisées pour son implémentation vont être exposées.

#### a) Permutation et expansion

La permutation consiste à réarranger les bits d'une chaîne. L'expansion est une forme spéciale de permutation qui réalise en plus la duplication des bits. Ces deux opérations ne nécessitent aucune ressource logique. Elles peuvent seulement être implémentées par câblage. Les sorties du bloc logique précédent sont câblées dans un ordre différent à l'entrée du prochain bloc logique.
Si la permutation est directement liée à l'entrée ou à la sortie du composant, comme c'est le cas de la permutation initiale et la permutation finale, le réarrangement prend place entre des broches d'entrés/sorties du composant et les blocs logiques aux quels ils sont connectés. Par conséquent, la permutation ou l'expansion ne cause aucun délais à l'exception de quelques retards dus à des liaisons complexes.
Dans notre conception, ces deux opérations ont été décrites en VHDL.
L'exemple d'une description en langage VHDL de la fonction PC1 de l'algorithme DES, où PC1 étant la permutation initiale de la clef, est donnée ci-après.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY pc1 IS
   PORT
    (
      key  : IN STD_LOGIC_VECTOR (0 TO 63);
     c0x   : OUT STD_LOGIC_VECTOR (0 TO 55)
    );
END pc1;

ARCHITECTURE behavior OF pc1 IS
SIGNAL xx : STD_LOGIC_VECTOR (0 TO 55);

BEGIN
```

```
xx(0) <= key(56); xx(1) <= key(48); xx(2) <= key(40); xx(3) <= key(32);
xx(4) <= key(24); xx(5) <= key(16); xx(6) <= key(8); xx(7) <= key(0);
xx(8) <= key(57); xx(9) <= key(49); xx(10) <= key(41); xx(11) <= key(33);
xx(12) <= key(25); xx(13) <= key(17); xx(14) <= key(9); xx(15) <= key(1);
xx(16) <= key(58); xx(17) <= key(50); xx(18) <= key(42); xx(19) <= key(34);
xx(20) <= key(26); xx(21) <= key(18); xx(22) <= key(10); xx(23) <= key(2);
xx(24) <= key(59); xx(25) <= key(51); xx(26) <= key(43); xx(27) <= key(35);
xx(28) <= key(62); xx(29) <= key(54); xx(30) <= key(46); xx(31) <= key(38);
xx(32) <= key(30); xx(33) <= key(22); xx(34) <= key(14); xx(35) <= key(6);
xx(36) <= key(61); xx(37) <= key(53); xx(38) <= key(45); xx(39) <= key(37);
xx(40) <= key(29); xx(41) <= key(21); xx(42) <= key(13); xx(43) <= key(5);
xx(44) <= key(60); xx(45) <= key(52); xx(46) <= key(44); xx(47) <= key(36);
xx(48) <=key(28); xx(49) <= key(20); xx(50) <= key(12); xx(51) <= key(4);
xx(52) <= key(27); xx(53) <= key(19); xx(54) <= key(11); xx(55) <= key(3);

c0x <= xx(0 TO 55);

END behavior;
```

**b) Registres**

Les registres (buffers de données) peuvent être implémentés soit en logique combinatoire, soit en utilisant des éléments RAM. La plupart des composants FPGAs récents comportent des éléments mémoire où l'implémentation de ces registres est plus efficace.

**c) Multiplexeurs**

Les multiplexeurs peuvent simplement être implémentés en utilisant la logique combinatoire. Pour leur implémentation, les outils de synthèse vont tenter d'utiliser des fonctions prédéfinies du composant FPGAs.

**d) Fonctions logiques standards**

Les fonctions logiques standards tel que le ET, le OU et le XOR sont à base de portes logiques élémentaires. Leur performance ne dépend pas de la largueur de la chaîne de bits sur laquelle ils opèrent. A titre d'exemple, dans le réseau de Feistel le XOR 32 s'exécute avec les mêmes performances que les XOR 48 bits.

**e) Tables–S**

Les tables–S sont des LUTs (Look-Up Tables) de taille égale à 6 x 4 bits et par conséquent, elles contiennent 64 valeurs de 4 bits (voir paragraphe I.4.1). L'implémentation des tables–S est décisive pour une conception efficace du DES [33]. Si elles sont implémentées via la logique combinatoire, elles nécessiteraient des centaines d'éléments logiques. Une étude par Grey Haskins [33] montre que l'utilisation d'éléments mémoire reste la méthode la plus efficace pour l'implémentation des tables–S.

**f) Registres à décalage**

Les registres à décalage, utilisés dans la diversification de la clef, peuvent être classés en circuits à décalage combinatoires et en circuits à décalage commandés. La figure II.3.2 montre une vue d'ensemble de ces deux types de circuits. Tous ces circuits font au plus une rotation circulaire d'un bit sur une chaîne donnée.

***Circuit à décalage combinatoire*** décale toujours avec un nombre fixe de positions indépendamment de l'horloge. Ils sont essentiellement à base de permutations (voir figure II.3.2). Ces circuits sont représentés par les blocs fonctionnels Ga_rot dans la figure II.3.1.

***Circuit à décalage commandé*** possède une entrée supplémentaire avec laquelle on décide si la donnée doit être décalée ou non. Un circuit à décalage commandé est à base de multiplexeur (voir figure II.3.2). Ces circuits sont représentés par les blocs fonctionnels Gm_rot dans la figure II.3.1.

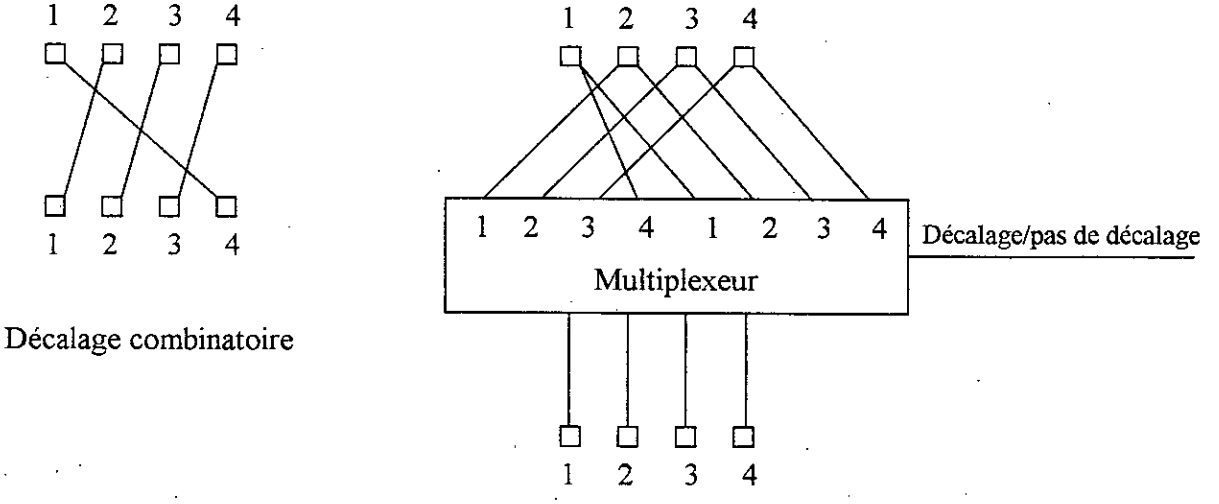



Figure II.3.2 : Implémentation des registres à décalage

### II.3.3 Optimisation de la conception

Une simple méthode pour l'optimisation de notre conception est l'utilisation des modules LogiBLOX. Ces modules LogiBLOX sont préconfigurés et optimisés pour les composants FPGAs de Xilinx (pour plus de détails voir paragraphe II.4.4). Tout les blocs fonctionnels réalisés à base de modules LogiBLOX sont montrés dans la figure II.3.1 avec un arrière plan hachuré.
La figureII.3.1 montre le schéma–bloc détaillé de l'architecture standard de DES. Elle contient tous les blocs fonctionnels déjà étudiés dans les sections précédentes. Cette conception a été implémentée sous l'appellation DES_VER1.

### II.3.4 Circuit du chemin de données

Le circuit du chemin de données a été conçu selon l'approche Down-Top (du Bas vers le Haut). Comme son nom l'indique, l'approche Down-Top consiste à faire une configuration qui commence du niveau le plus bas jusqu'à atteindre la structure supérieure.
Cette approche a été adoptée par le fait que la structure de base de l'algorithme DES est constituée de plusieurs sous-blocs (voir paragraphe II.2.2) ; un sous-bloc est constitué à son tour de plusieurs blocs fonctionnels et ainsi de suite.

#### II.3.4.1 Circuit de diversification de la clef

Il est possible d'inclure la logique concernant la partie déchiffrement en consommant plus de ressources du composant FPGAs, mais en respectant toujours les mêmes contraintes en temps. Tel que décrit dans le paragraphe I.4.3, le déchiffrement signifie que nous devons faire un décalage à droite de la clef, soit zéro , une , ou deux fois.

Afin de réaliser les deux opérations de chiffrement et de déchiffrement, la clef traverse deux branches (voir figure II.3.3). Dans la première branche, elle subit un décalage à gauche d'une ou de deux positions suivant la valeur de l'itération. Cette opération est nécessaire pour le chiffrement. Dans la seconde branche, la clef subit un décalage à droite de zéro, une, ou de deux positions suivant la valeur de l'itération. Cette dernière opération est nécessaire pour un déchiffrement.

Les deux opérations de chiffrement et de déchiffrement sont ainsi réalisées en même temps et l'ensemble des circuits de décalages ne sont commandés que par deux signaux de contrôle (SFT1 et SFT2).
A la fin de ce circuit, un multiplexeur, commandé par le signal de sélection de l'opération chiffrement ou déchiffrement, commute entre le résultat des deux branches; ainsi il commute entre les deux opération, chiffrement ou déchiffrement.

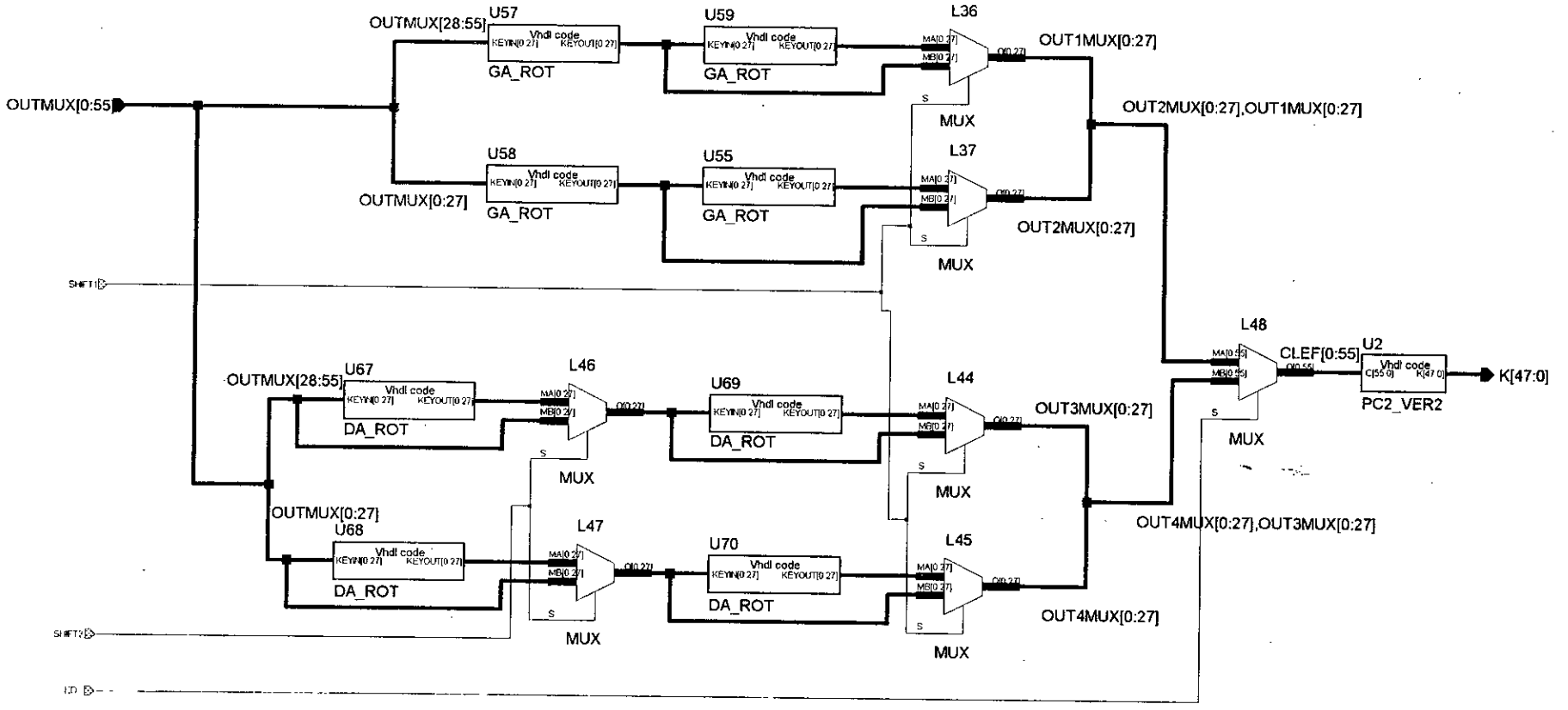



Figure II.3.3 : Circuit de diversification de la clef

### II.3.4.2 Circuit du réseau de Feistel

Le circuit du réseau de Feistel est représenté par la figureII.3.4. Sur ce schèma, on retrouve les quatre blocs fonctionnels constituants ce circuit, à savoir :

- Le bloc d'expansion (XP) décrit en langage VHDL,
- Les fonctions logiques (XOR) conçues à base de module LogiBLOX,
- Les tables–S (SBOXES) conçues à base de modules mémoire et la permutation P décrite en VHDL sont insérées dans le circuit du réseau de Feistel sous forme de module (module H10). Ces deux blocs fonctionnels sont représentés par la figure II.3.5.

### II.3.4.3 Circuit du chemin de données

Les circuits du chemin de données correspondants aux différentes conceptions des versions architecturales de l'algorithme DES (voir tableau II.2.1) sont données ci-dessous :

#### a) Conception DES_VER1

Cette conception correspond à l'architecture en structure de base de l'algorithme DES. Les éléments constituants le chemin de données de cette conception sont montées dans le circuit de la figure II.3.6 où nous retrouvons les différents blocs fonctionnels :

- circuits de permutation initiale (PC1_BIS et IP64),
- multiplexeurs (MUX),
- circuit de l'unité logique combinatoire (CLU),
- registres (DATA_REG),
- circuit de permutation finale (FP).

#### b) Conception DES_VER2.1

Cette version correspond à la conception de l'algorithme DES ayant une architecture en boucle expansée deux (02) fois. Cette conception est montrée dans la figure II.3.7. Elle contient tout les blocs fonctionnels de la conception DES_VER1 plus la duplication de l'unité logique combinatoire.

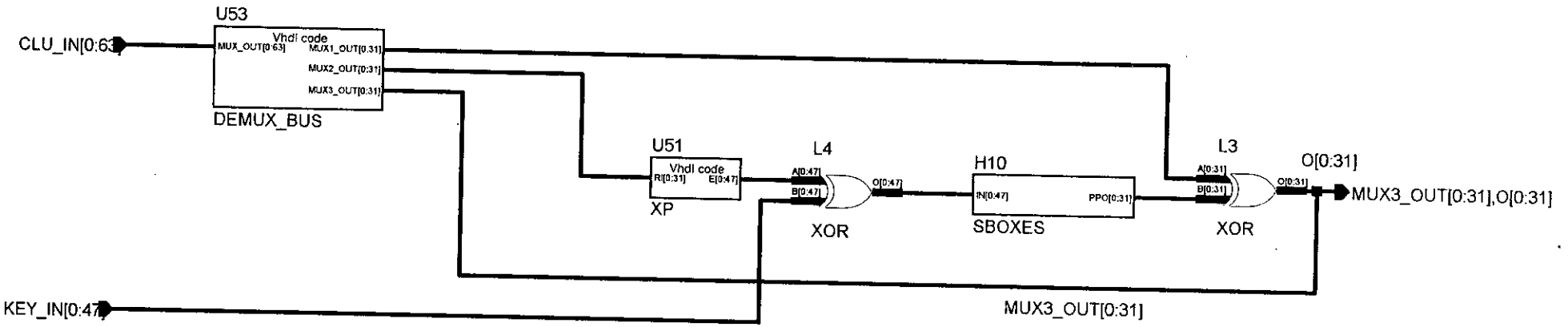



Figure II.3.4 : Circuit du réseau de Feistel

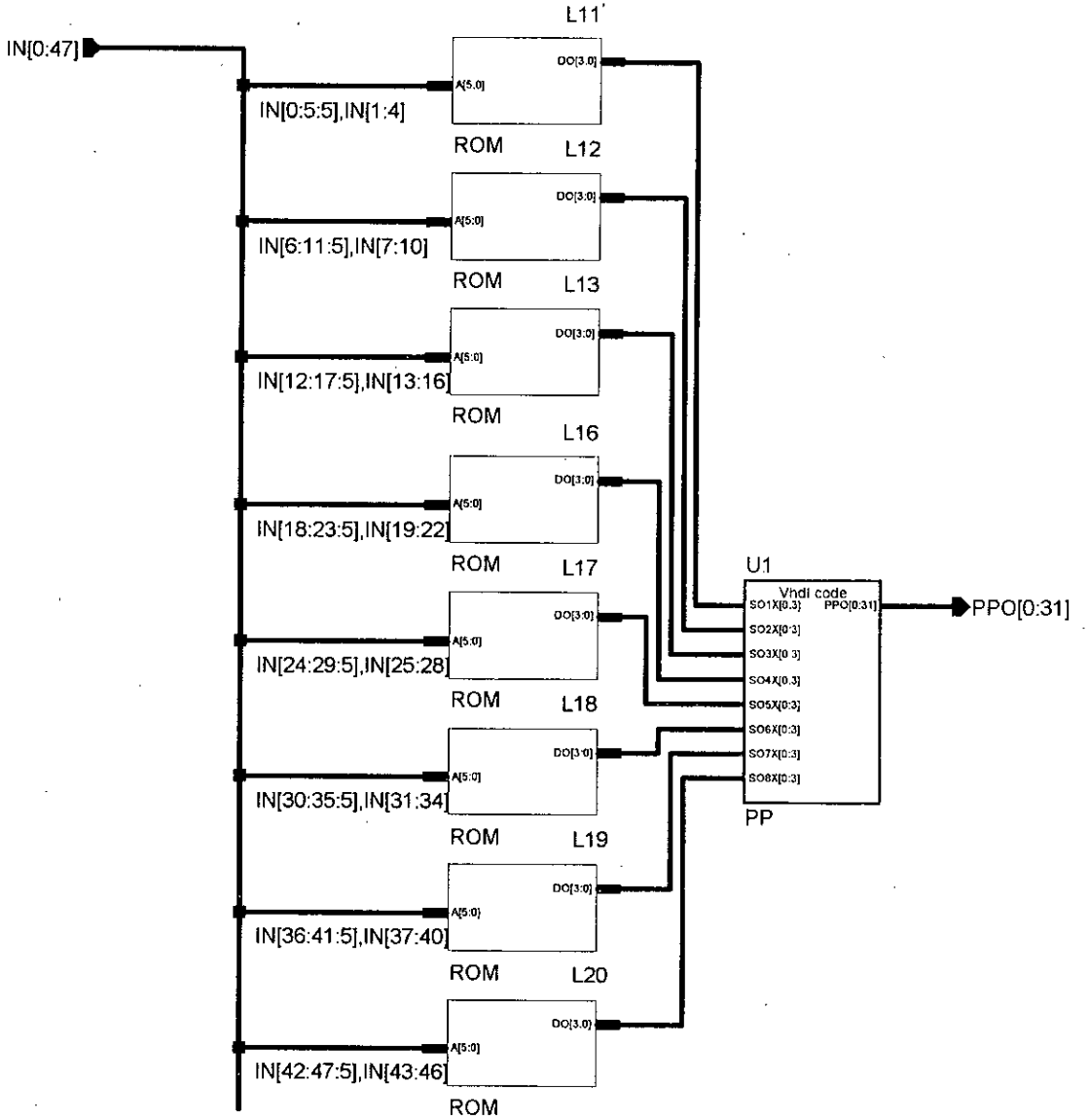



Figure II.3.5 : Circuit des Tables-S et Permutation P

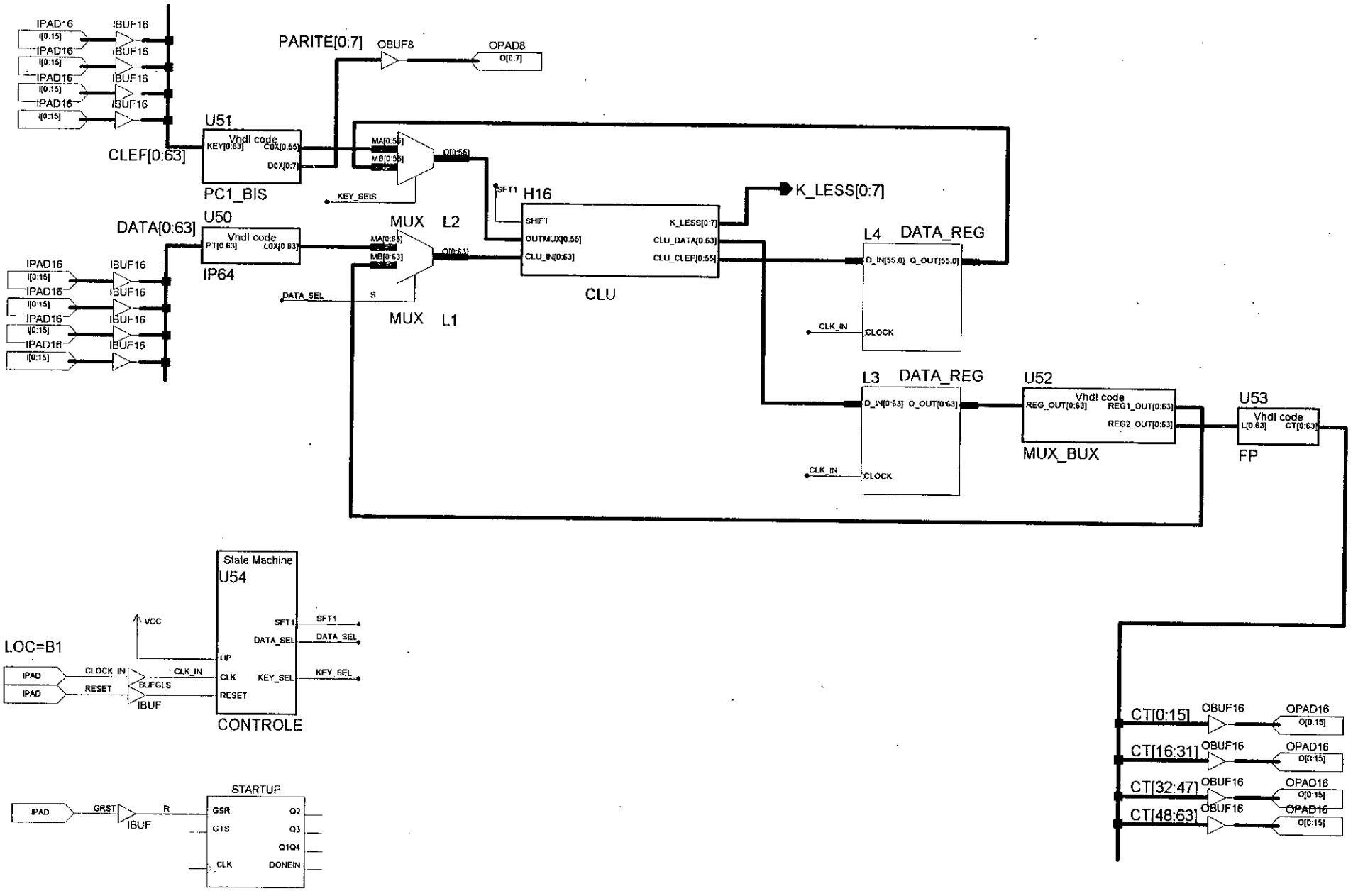



Figure II.3.6 : Circuit de la conception DES_VER1 à architecture standard

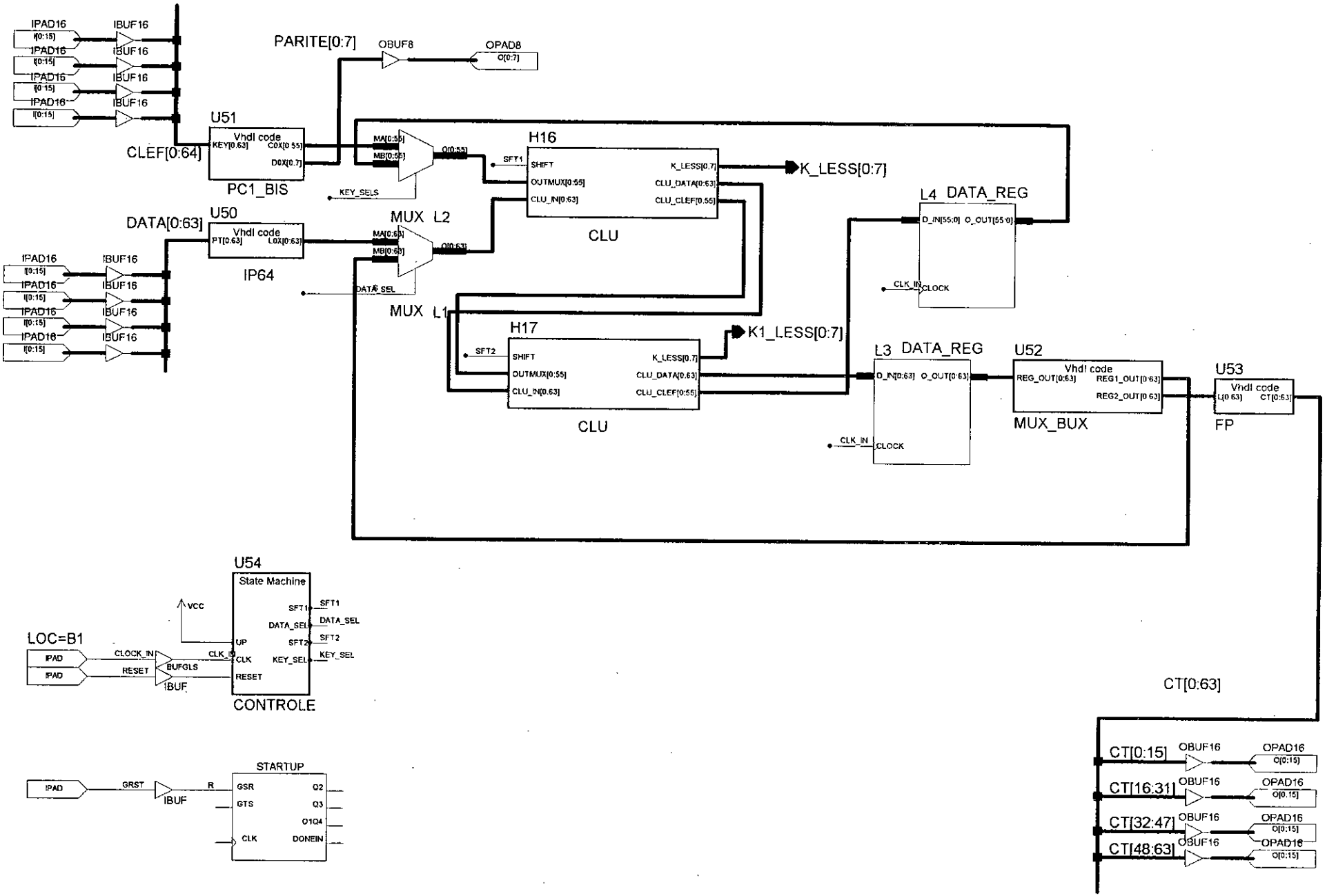



Figure II.3.7 : Circuit de la conception DES_VER2.1 à architecture en boucle expansée

**c) Conception DES_VER2.2**

Cette conception est donnée dans la figure II.3.8. La différence par rapport à la conception de l'algorithme DES en structure de base est l'existence de quatre unités logiques combinatoire montées en cascade.

**d) Conception DES_VER3.1**

C'est la conception de l'algorithme DES ayant une architecture en pipeline. Elle est similaire à la conception avec une structure en deux boucles expansées. Sauf l'addition des registres (L9 et L10) entre les deux unités combinatoire (CLU). Ceci est montré dans la figure II.3.9.

**e) Conception DES_VER3.2**

Le circuit du chemin de données de cette conception est montré par la figure II.3.10. Il possède quatre pipelines et composé essentiellement des blocs fonctionnels suivants :

- circuits de permutation initiale (PC1_BIS et PC2),
- multiplexeurs (MUX),
- quatre circuits de l'unité logique combinatoire (CLU),
- registres (DATA_REG),
- circuit de permutation finale (FP).

**f) Conception DES_VER4**

C'est la conception qui possède une architecture combinée. Le chemin de données de cette conception est constituée essentiellement de quatre unités combinatoires séparées en deux par les registres (L9 et L10). Le circuit du chemin de données et montré par la figure II.3.11.

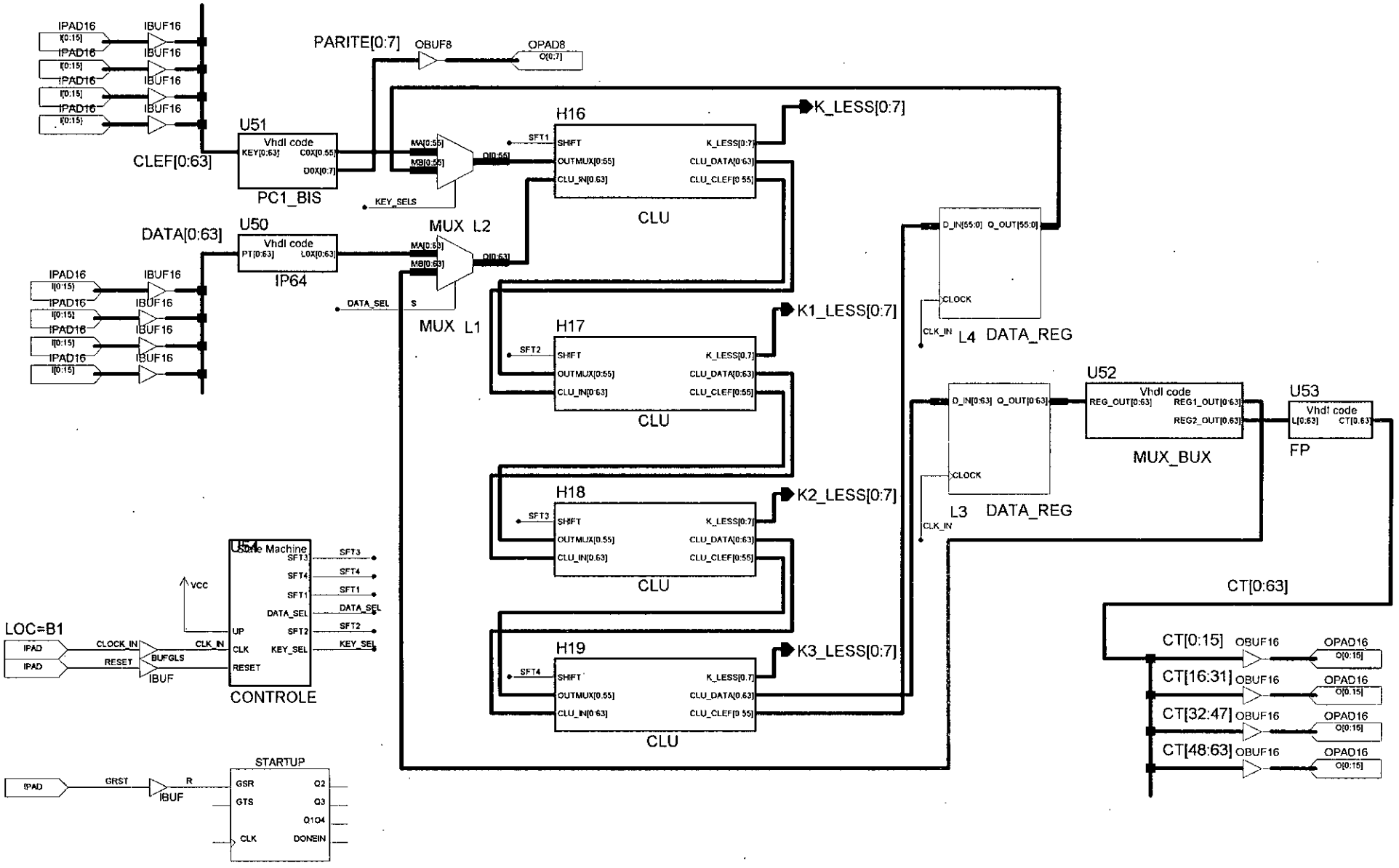



Figure II.3.8 : Circuit de la conception DES_VER2.2 (4 expansions de la boucle)

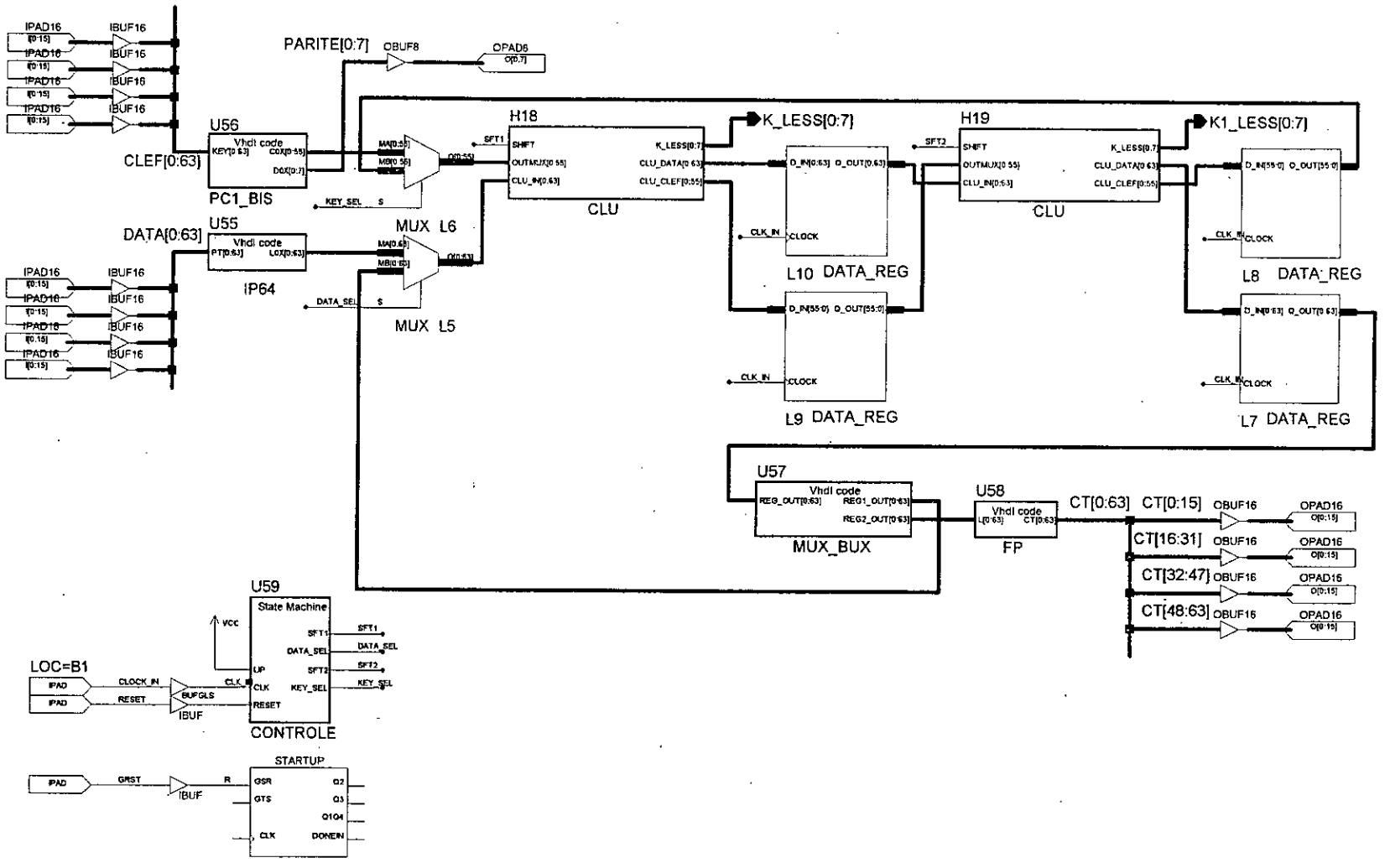



Figure II.3.9 : Circuit de la conception DES_VER3.1 (2 pipelines)

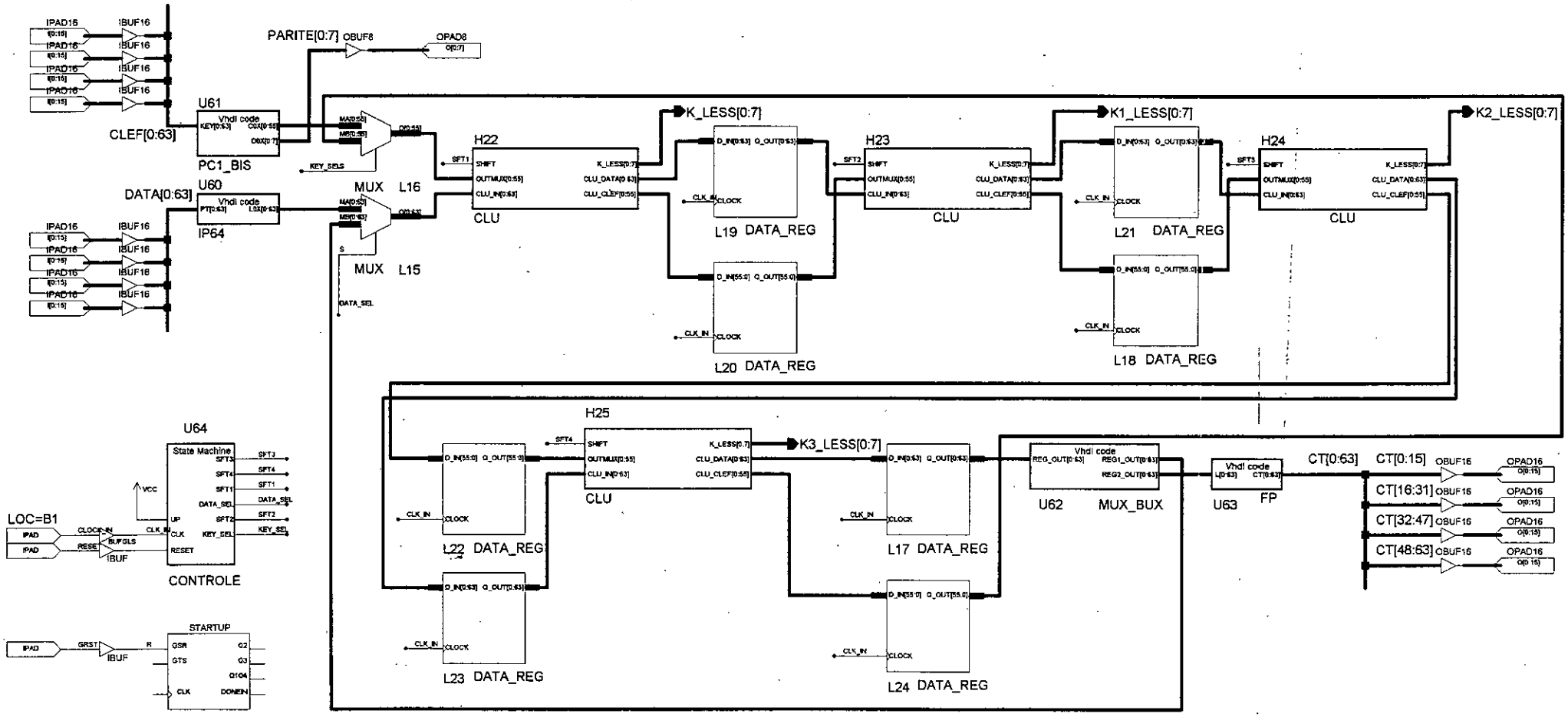



Figure II.3.10 : Circuit de la conception DES_VER3.2 (4 pipelines)

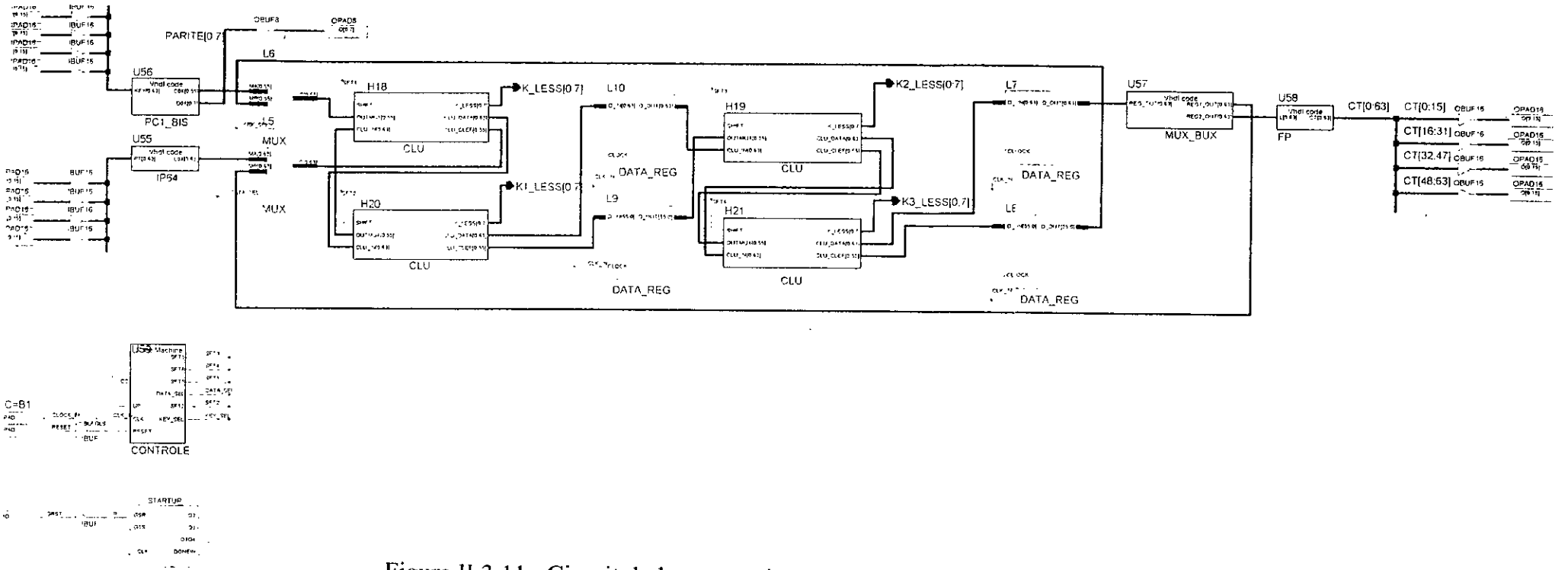



Figure II.3.11 : Circuit de la conception DES_VER4 (structure combinée)

### II.3.5 Logique de contrôle

La logique de contrôle du chemin de données qui vient d'être décrit a été conçue sous forme de machine d'état (voir figure II.3.12). Le calcul du texte chiffré pour une conception du DES en structure de base nécessite 16 itérations consécutives donc la machine d'état correspondante doit comprendre 16 états ordonnés et disposés suivant une seule boucle. La transition d'un état à l'autre est déclenchée par le signal d'horloge. Un signal de validation de l'horloge est implémenté pour permettre d'arrêter la machine d'état à n'importe quel cycle d'horloge. Le signal RESET, appliqué sur un des états de la machine provoque le retour à l'état initial.

Le nombre d'état de la machine d'état varie selon le nombre d'itération (voir tableau II.2.1) requit par chaque conception de l'algorithme DES.

Les signaux de commande que le machine d'état fournie sont :

- **Data_Sel** : commande l'entrée du multiplexeur, du chemin de données, pour le chargement d'une nouvelle donnée ou bien d'une donnée provenant de la boucle.

- **Key_Sel** : commande l'entrée du multiplexeur, de la partie diversification de la clef, pour le chargement d'une nouvelle clef ou bien d'une clef provenant de la boucle.

- **SFT1** : signal de commande des registres à décalage.

- **SFT2** : signal de commande des registres à décalage.

- **ED** : sélection entre une opération de chiffrement et opération de déchiffrement.

D'autres signaux de commandes sont nécessaires pour chaque conception de l'algorithme DES. Ces signaux sont détaillés dans le sous-chapitre traitant de l'implémentation du modèle structurel associé à l'algorithme DES.

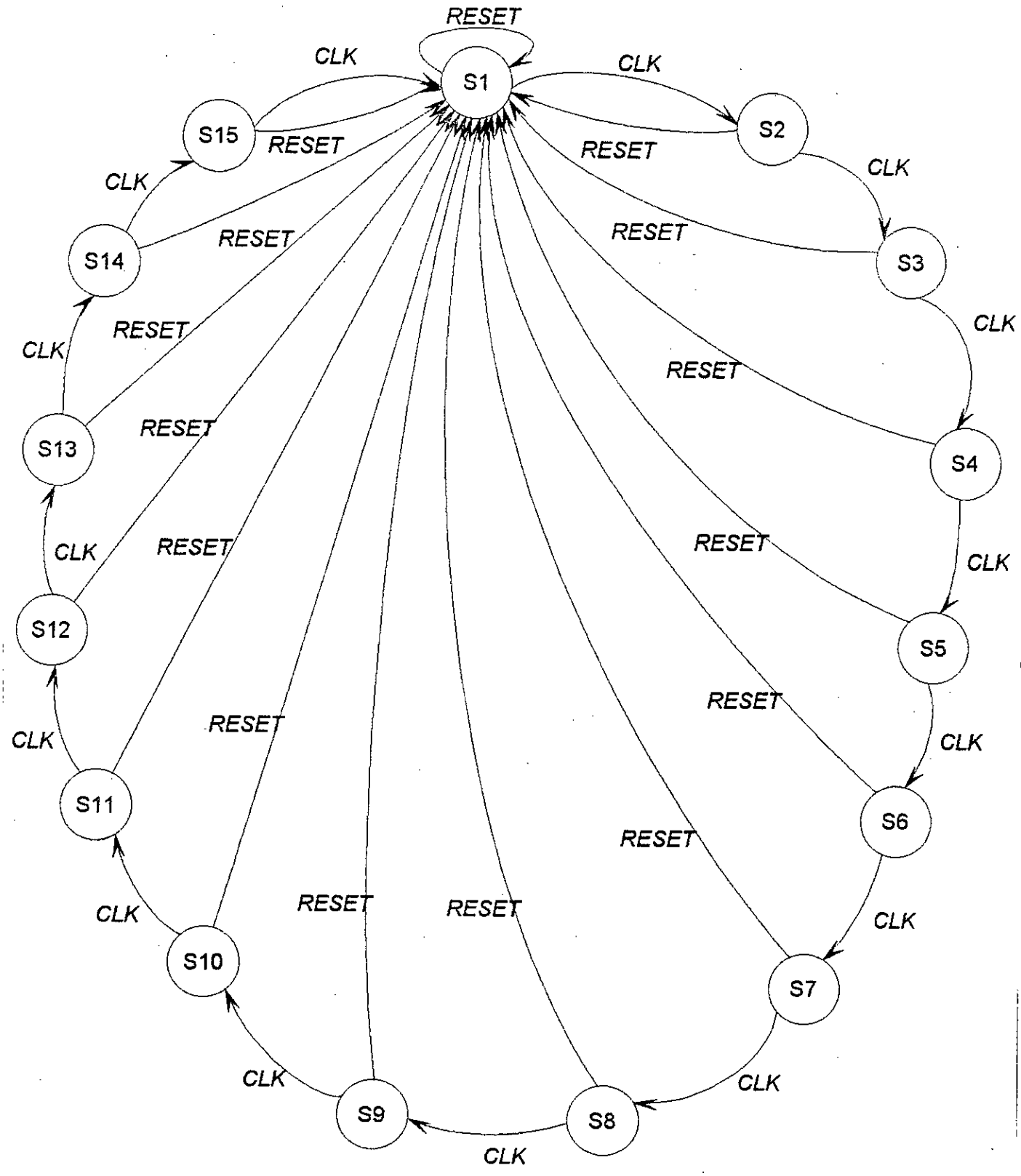



Figure II.3.12 : Machine d'état du contrôleur

### II.3.6 Conclusion

Afin d'arriver à réaliser une implémentation, la plus performante que possible de l'algorithme DES nous devons respecter les recommandations suivantes :

- Les tables-S doivent être implémentées sur des modules ROM. La rapidité d'exécution des tables-S influe considérablement sur la performance de la conception globale.

- Les blocs fonctionnels de permutation et d'expansion doivent être implémentés en utilisant seulement les ressources d'interconnexions.

- Les registres à décalage peuvent être implémentés en utilisant uniquement des ressources d'interconnexions associées à des multiplexeurs (éléments disponibles dans la structure de base d'un CLB).

- L'utilisation des modules LogiBLOX, optimisés selon le composant FPGA cible, facilitent considérablement l'opération de conception.

# II.4 - Implémentation du modèle structurel associé à l'algorithme DES

## II.4.1 Introduction

Nous avons implémenté diverses versions architecturales de l'algorithme DES (voir tableau II.2.1). Chacune de ces architectures est implémentée sur un composant spécifique.

## II.4.2 Choix du composant

Nous avons choisi des composants FPGAs de Xilinx pour notre implémentation. Cette décision est basée sur les travaux de recherche décrits en [33]. Le résultat le plus important apporté par cette étude consiste en la difficulté d'implémenter plus d'un ensemble de tables–S sur un composant reconfigurable disponible sur le marché, tel que, les composants EPLDs d'Altera alors, que plusieurs ensembles de tables–S sont nécessaires pour l'implémentation d'une architecture en boucle expansée ou en pipeline de l'algorithme DES.

## II.4.3 Fichiers sources en VHDL

Pour certains blocs fonctionnels des fichiers VHDL ont été crées séparément, à l'exception de ceux qui utilisent les LogiBLOX. Le tableau II.4.1 liste des différents fichiers VHDL et leurs fonctions respectives.

| Nom de fichier | Fonction |
|---|---|
| IP.vhd | Permutation initiale |
| FP.vhd | Permutation finale |
| PC1.vhd | Permutation PC-1 |
| PC2.vhd | Permutation PC-2 |
| XP.vhd | Fonction d'expansion E |
| PP.vhd | Permutation P |
| Ga.vhd | Unité combinatoire de décalage à gauche |
| Da.vhd | Unité combinatoire de décalage à droite |

Tableau II.4.1 : Fichiers source VHDL et leurs fonctions

### II.4.4 LogiBLOX

La majorité des blocs fonctionnels, tels que, les registres, les multiplexeurs, les tables–S ont été créés avec l'outil LogiBLOX. Le tableau II.4.2 liste les modules crées avec cet outil et leurs fonctions respectives. Ces modules LogiBLOX sont préconfigurés et optimisés pour les composants FPGAs de Xilinx. Leur performance ne dépend pas de la qualité des outils de synthèse, contrairement aux modules décrits en VHDL qui nécessitent des outils de synthèse très performants.
L'outil graphique et interactive LogiBLOX, créé des fichiers *.*ngc* qui seront utilisés par les autres outils, par exemple, le Xilinx design manager, le simulateur. ...etc.

| **Nom de fichiers** | **Fonctions** |
|---|---|
| Mux64 | Multiplexeur 64 bits |
| Mux56 | Multiplexeur 56 bits |
| Mux28 | Multiplexeur 28 bits |
| Reg64 | Registre 64 bits |
| Reg56 | Registre 56 bits |
| Rom1 | Table–S 1 |
| Rom2 | Table–S 2 |
| Rom3 | Table–S 3 |
| Rom4 | Table–S 4 |
| Rom5 | Table–S 5 |
| Rom6 | Table–S 6 |
| Rom7 | Table–S 7 |
| Rom8 | Table–S 8 |

Tableau II.4.2 : Modules LogiBLOX et leurs fonctions respectives.

### II.4.5 Implémentation de la conception

Nous avons implémenté le DES en plusieurs versions afin de pouvoir comparer les performances des différentes architectures. Toutes les conceptions ont été implémentés sur le composant **XC4013BG256-3** du fait que le logiciel *Xilinx Foundation Series 1.5, Student Version* offre une liste de composants dont les performances ne dépassent pas le composant FPGA **XC4013BG256-3**.

Le fait d'être limiter par le type de composants FPGAs disponibles dans le logiciel *XILINX Foundation Series 1.5 Student version* et pour pouvoir implémenter toutes les conceptions et ainsi comparer leurs performances. Nous avons restreint nos différentes conceptions à supporter uniquement l'opération de chiffrement.
Seule une version de la conception DES_VER1 nommée DES_VER1.1 supporte les deux opérations chiffrement et déchiffrement. La conception DES_VER1.1 sera utilisée dans le bloc de chiffrement/déchiffrement du cryptoprocesseur.

### II.4.5.1 DES_VER1

C'est la toute première conception que nous avons implémentée. Une opération de chiffrement nécessite 16 cycles d'horloges et aucune technique d'expansion de la boucle ou de pipeline n'a été utilisée. DES_VER1 supporte uniquement l'opération de chiffrement.

Le schéma de la version DES_VER1 est illustré par la figure II.3.1. Le composant cible est le circuit **XC4013XLBG256-3**.

La logique de contrôle pour cette conception doit fournir les signaux suivants : data_sel, key_sel, et SFT (pour une description des signaux voir chapitre 6.5). La figure II.4.1 montre le diagramme temporel de cette conception. Les nombres inscrits sur le signal SFT1 représente la sous-clef calculée pendant l'état respective. Pendant l'état E1 il y a génération de la première sous-clef, pendant l'état E2 la seconde sous-clef, et ainsi de suite. Pendant l'état E16 on calcul la première sous- clef de la prochaine clef qui est indiquée par une couleur gras sur le graphe.

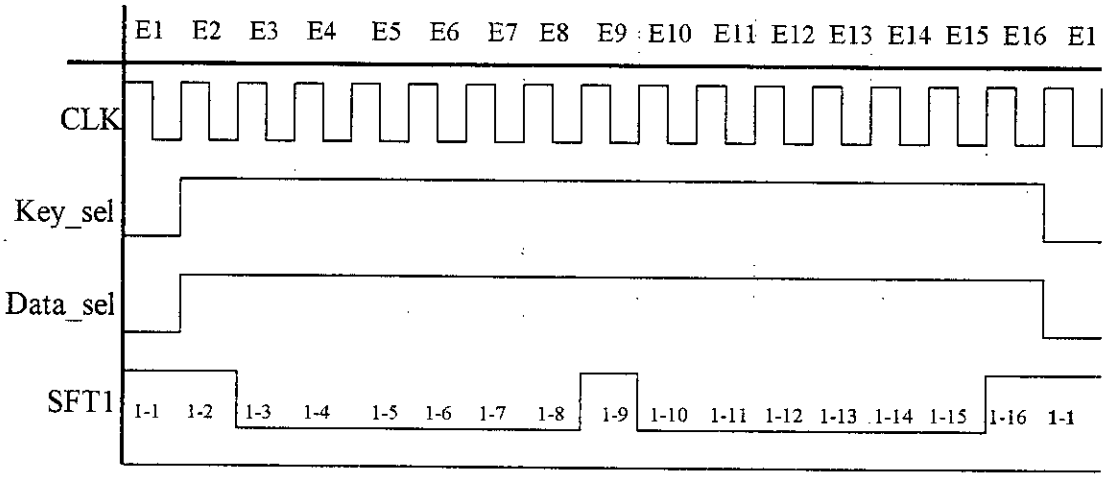



Figure II.4.1 : Signaux de contrôle pour la version DES_VER1

### II.4.5.2 DES_VER1.1

L'implémentation DES_VER1.1 est une version de l'implémentation DES_VER1, la seule différence entre les deux versions est que le première supporte les opérations de chiffrement et de déchiffrement.
Cette version a été elle aussi implémentée sur le composant FPGA **XC4013BG256-3**. Sa logique de contrôle dont les signaux de contrôles sont représentés par la figureII.4.2 possède un signal supplémentaire SFT2 dont le rôle et de commander les deux premiers circuits de décalage de la partie déchiffrement du circuit de diversification de la clef (voir figure II.3.3).

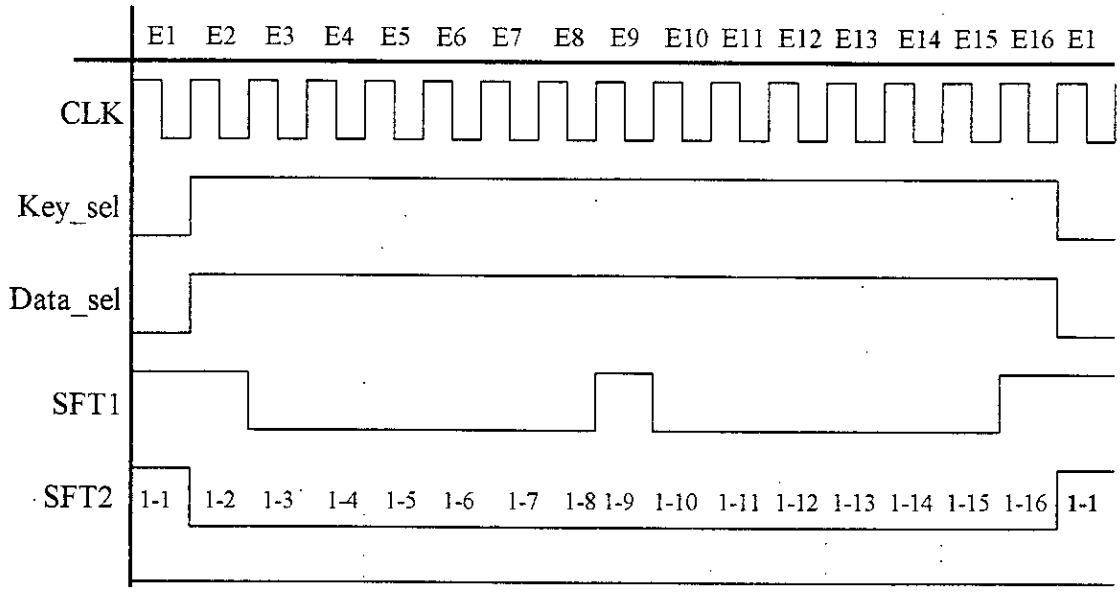



Figure II.4.2 : Signaux de contrôle pour la version DES_VER1.1

### II.4.5.3 DES_VER2.1

C'est la première conception à base d'une architecture en boucle expansée. Une opération de chiffrement prend 8 cycles d'horloge. DES_VER2.1 a pour circuit cible, le composant **XC4013BG256-3** dont ses ressources suffisent largement.

La logique de contrôle possède un signal additionnel SFT2. Il a fondamentalement la même fonction que le signal SFT1 mais son action porte sur la seconde partie du circuit de diversification de la clef. La figure II.4.3 montre le diagramme temporel. Durant chaque état il y a calcul d'un bloc de donnée et la sous–clef qu'il lui est associée. Après les huit cycles d'horloge une nouvelle paire donnée–clef est chargée. Elle est montrée en couleur gras sur le diagramme.

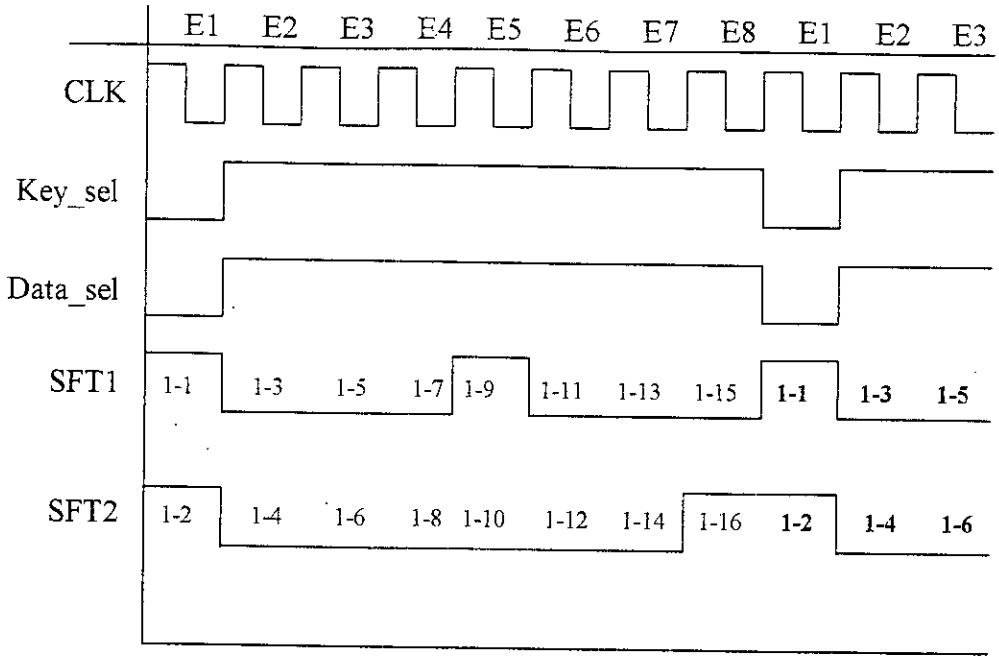



Figure II.4.3 : Signaux de contrôle pour la version DES_VER2.1

### II.4.5.4 DES_VER2.2

DES_VER2.2 est la seconde conception à base d'une architecture en boucle expansée, mais cette version la boucle a subit quatre (04) expansions. Donc une opération de chiffrement prend 4 cycles d'horloge. La machine d'état possède uniquement 4 états. Cette conception est implémentée sur le circuit cible XC4013XL-3BG256.

La logique de contrôle possède trois signaux additionnels SFT2, SFT3 et SFT4. Ces signaux commandent la seconde, la troisième et la quatrième partie de diversification des clefs. La figure II.4.4 montre le digramme temporel de ces signaux de commande. Pendant chaque état il y a calcul d'un bloc de donnée et la sous-clef qu'il lui est associée. Après les quatre cycles d'horloge une nouvelle paire donnée-clef est chargée. Elle est montrée en couleur gras sur le diagramme.

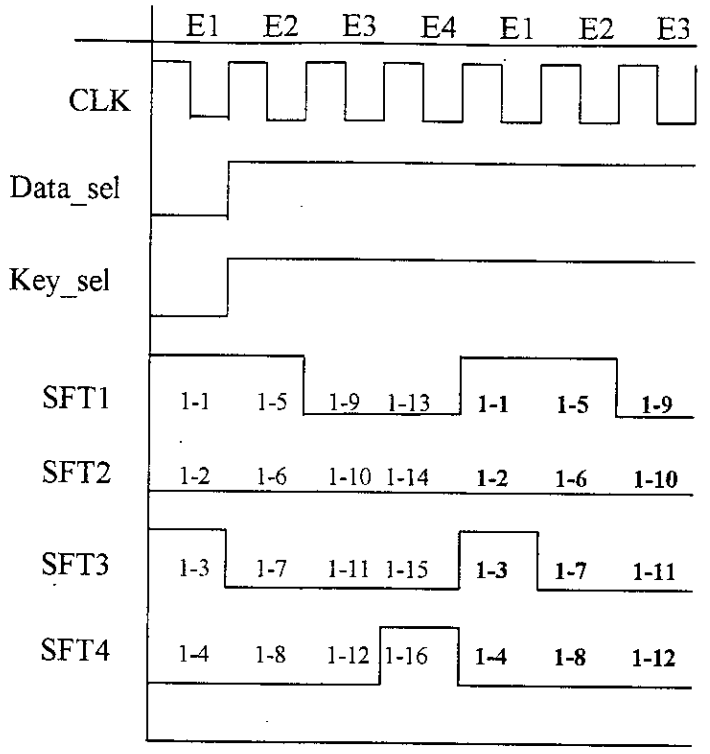



Figure II.4.4 : Signaux de contrôle pour la version DES_VER2.2

### II.4.5.5 DES_VER3.1

DES_VER3.1 est la première conception avec une structure en pipeline. Une opération de chiffrement prend 16 cycles d'horloges, deux opérations peuvent s'exécuter en même temps. DES_VER3.1 a été implémentée sur le circuit cible **XC4013XL-BG256-3**.

DES_VER3.1 possède un signal additionnel SFT2 qui opère sur la seconde partie de diversification des clefs. La figure II.4.5 montre le diagramme temporel. Comme c'est une conception en pipeline, elle peut travailler sur 2 blocs de données en même temps, d'où la notation 2-16, sur le signal SFT2, qui représente la $16^{ème}$ sous–clef du second bloc de la clef. La première paire donnée–clef est chargée pendant l'état E1, alors que la seconde paire est chargée à chaque état E2.
Durant le prochain état E1 la $16^{ème}$ sous–clef du second bloc de donnée est généré par la seconde partie de diversification des clefs, tandis que la première partie de diversification des clefs calcul la $1^{ère}$ sous–clef du nouveau bloc de donnée, et ainsi de suite jusqu'à la fin de chiffrement du message.

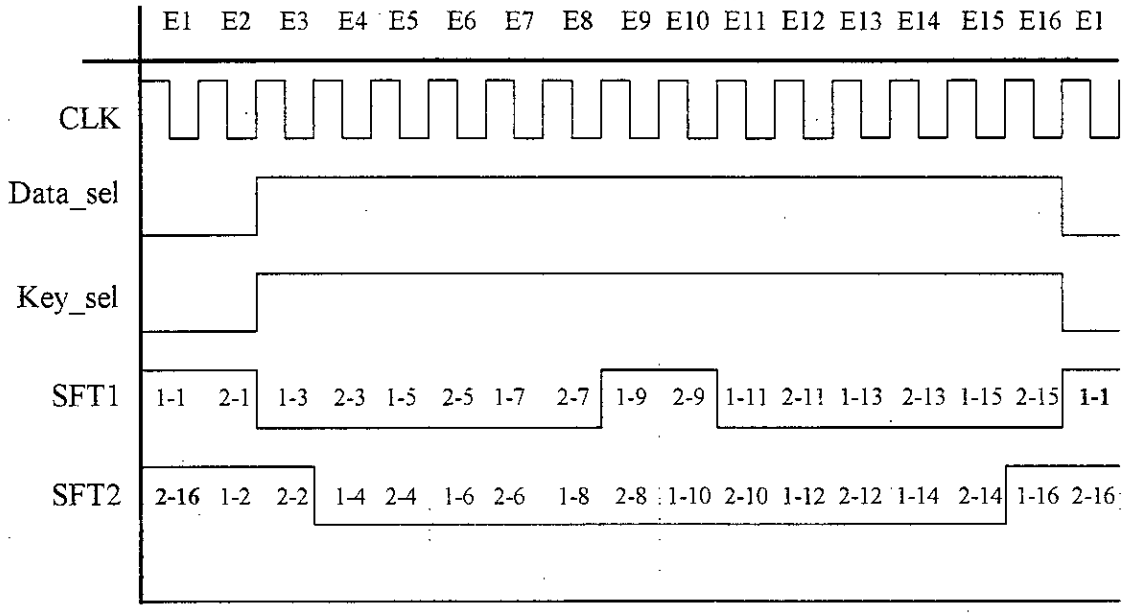



Figure II.4.5 : Signaux de contrôle pour la version DES_VER3.1

### II.4.5.6 DES_VER3.2

C'est la seconde conception à base d'une architecture en pipeline, cette dernière comprend 4 pipelines, permettant ainsi à quatre (04) opérations de s'exécuter en même temps et chaque opération de chiffrement dure 16 cycles d'horloge. DES_VER3.2 a été implémentée sur le composant XC4013XL-3BG256.

La logique de contrôle fournit trois signaux additionnels SFT2, SFT3 et SFT4 opérant sur la seconde, la troisième et la quatrième partie diversification des clefs. Ces signaux de commande sont représentés sur le diagramme temporel de la figure II.4.6.
La première paire donnée–clef est chargée pendant l'état E1, suivie de la seconde paire durant l'état E2 et ensuite il y a chargement des deux autres paires données–clefs, respectivement pendant les états E3 et E4 jusqu'au remplissage du pipeline pendant quatre (04) cycles d'horloge.

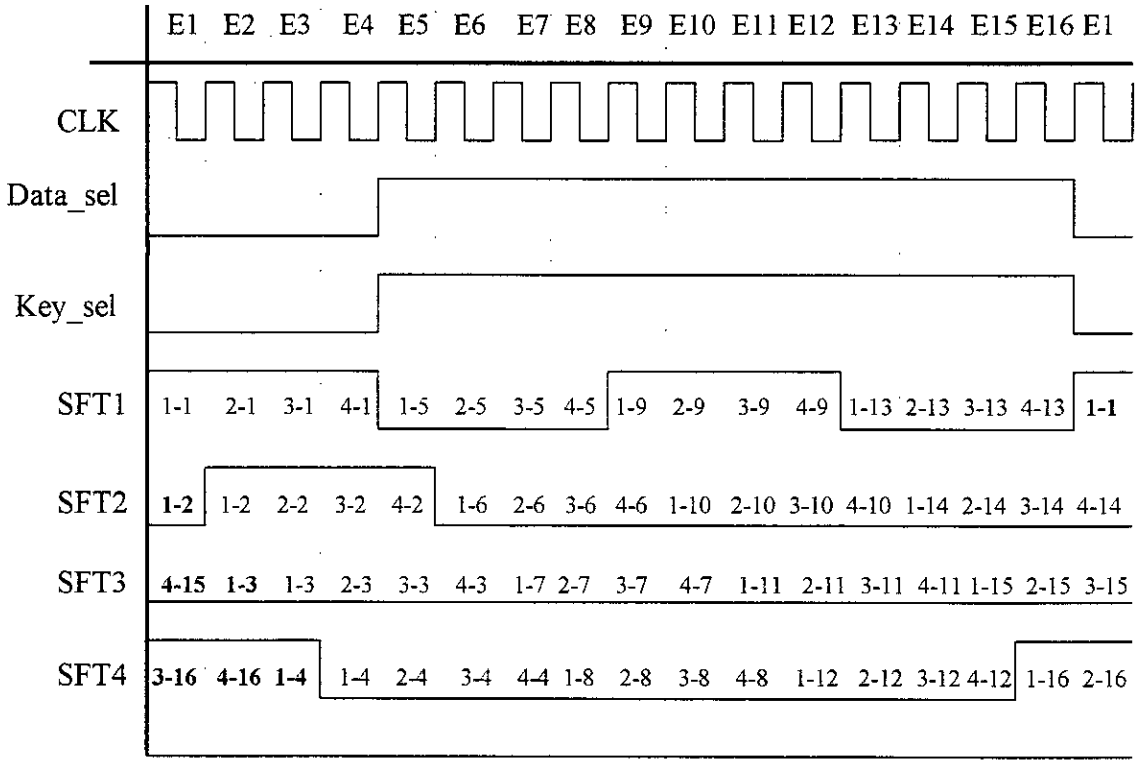



Figure II.4.6 : Signaux de contrôle pour la version DES_VER3.2

### II.4.5.7 DES_VER4

Cette conception est une combinaison des architectures en boucle expansée et en pipeline. Elle contient deux pipelines avec chacun deux structures en boucle expansée. Chaque opération de chiffrement prend 8 cycles d'horloge, deux opérations peuvent être traitées en même temps. DES_VER4 est aussi implémentée sur le composant **XC4013XLBG256-3**.
Le chargement des paquets clefs et des paquets données est similaire à la conception DES_VER3.1. Mais la différence réside dans le fait que le résultat est calculé pendant 8 cycles d'horloge et le prochain cycle de chargement commence juste après.
La logique de contrôle fournit trois signaux additionnels : ST2, ST3 et ST4 qui opèrent sur la seconde, la troisième et la quatrième partie de diversification des clefs. La figureII.4.7 montre le diagramme temporel de cette conception.

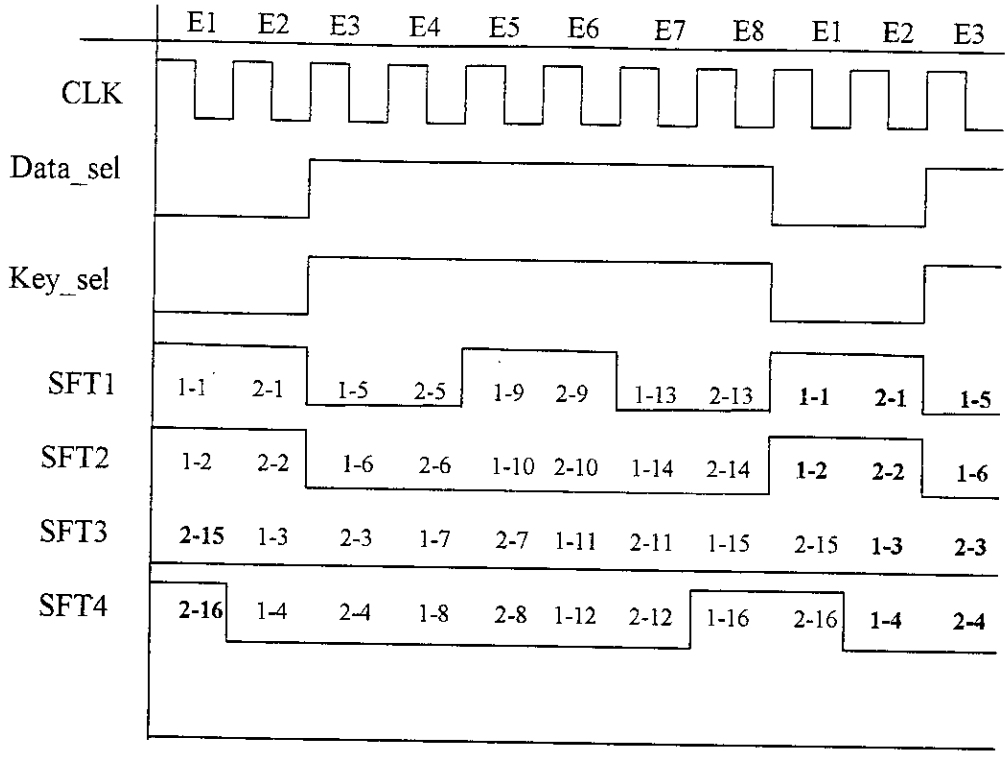



Figure II.4.7 : Signaux de contrôle pour la version DES_VER4

## II.4.6 Conclusion

Concernant l'architecture de l'algorithme DES avec une structure en boucle expansée, nous remarquons que plus il y a expansion de la boucle, plus nous diminuons le nombre de cycles de la machine d'état et en même temps nous augmentons le nombre des signaux de contrôle des circuits de diversification des clefs; sans pour autant trop accroître la complexité de la machine d'état.
Par contre, si nous augmentons le nombre de pipeline, nous augmentons forcément le nombre de signaux de contrôle nécessaires à la commande des circuits de diversification des clefs des différents étages du pipeline et surtout nous rendons plus complexe la machine d'état. Le nombre de cycle de la machine d'état reste toujours constant.

Les modèles physiques résultants de l'implémentation sur le circuit FPGA XC4013XL-3XL-3BG256 de l'algorithme DES et de ses divers options architecturales sont présentés respectivement dans les annexes A.1, A.2, A.3, A.4 et A.5.

Après la phase d'implémentation nous avons procédé à la simulation de l'implémentation de l'algorithme DES et de ses divers options architecturales afin de confirmer les performances données par l'analyseur temporel statique. Toutes ces simulations sont données dans annexes B.1, B.2, B.3, B.4, B.5 et B.6.

# Chapitre III

# Implémentation du cryptoprocesseur sur FPGA

## III.1 Introduction

L'implémentation du cryptoprocesseur sur un composant FPGA passera lui aussi par les trois domaines d'expression du modèle de conception.
L'étude des éventuelles optimisations architecturales et des éléments constituants le modèle comportemental du cryptoprocesseur notamment sa structure interne, son modèle de programmation, son fonctionnement globale et le séquencement des cycles fonctionnels élémentaires nécessaires au traitement d'une instruction permettront de dégager un modèle architectural du processeur (CISC ou RISC) ainsi qu'un modèle d'exécution (séquentiel ou parallèle).
Le modèle structurel présentera l'étude détaillée des composantes du cryptoprocesseur au travers des deux parties qui le constituent, à savoir, le chemin de données et le séquenceur. A cet effet, nous adopterons une approche de conception Down-Top qui facilitera considérablement notre travail et permettra de simuler chaque bloc fonctionnel séparément. En plus, nous réaliserons un module qui permettra d'interfacer entre le bloc de chiffrement/déchiffrement et les autres parties du cryptoprocesseur, en permettant ainsi à l'ensemble des composantes du cryptoprocesseur de travailler dans un environnement parfaitement synchronisé.
Nous terminerons ce chapitre par le choix d'une composant FPGA cible et ainsi l'implémentation du modèle structurel arrêté.

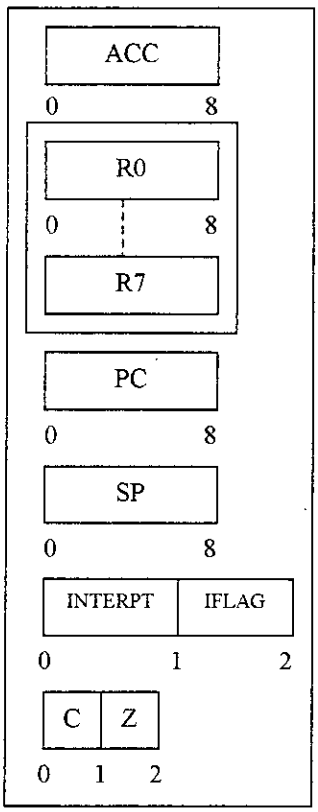
# III.2 - Modèle comportemental du cryptoprocesseur

## III.2.1 Introduction

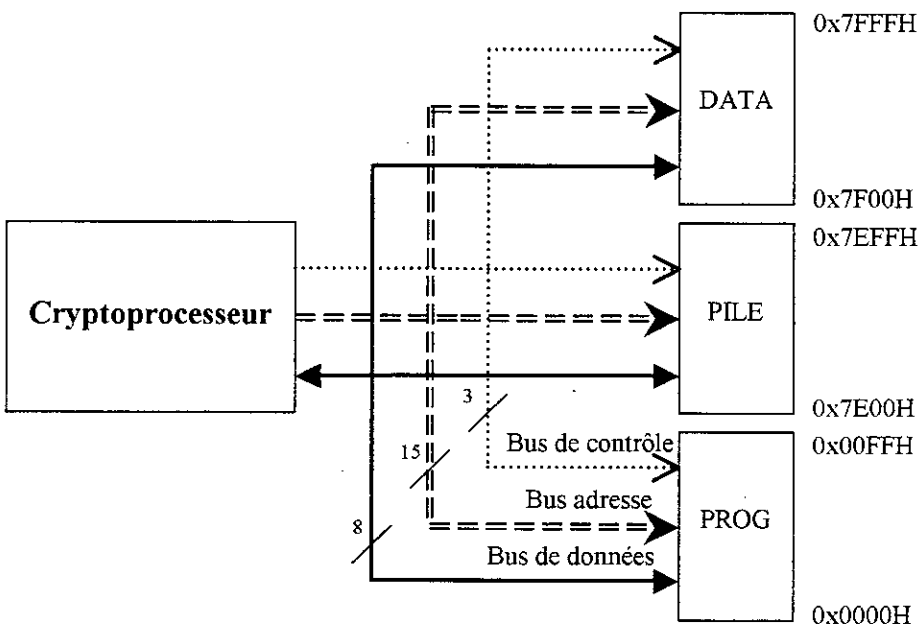
Notre travail consiste à concevoir un cryptoprocesseur qui réalise les fonctions essentielles d'un microprocesseur complétées par des fonctions spécialisées dans le domaine de la cryptographie. Ce qui feront de ce circuit un processeur dédié.
Ces fonctions s'expriment par une unité spécialisée dans le chiffrement et le déchiffrement. Elle sera réalisée à base d'un l'algorithme cryptographique préalablement choisi.
L'unité de chiffrement/déchiffrement doit travailler en parfaite synchronisme avec les autres unités du cryptoprocesseur et peut, dans certains cas, travailler à leur insu.
Quant à l'architecture du processeur pris isolement, sa structure découle de celle d'une architecture classique de VON NEUMANN.

## III.2.2 Modèle de programmation

La figure III.2.1.a représente le modèle de programmation au quel nous avons opté.




(a) Modèle de programmation




(b) Modèle mémoire

Figure II.2.1 : Choix du modèle de programmation et de la segmentation de la mémoire

Comme le montre la figure ci dessus, les éléments constitutifs de ce modèle sont:

- l'accumulateur (ACC) de 8 bits,
- l'ensemble de 16 registres (R0...R15) de 8 bits chacun,
- un compteur de programme (PC) de 8 bits,
- un pointeur de pile (SP) de 8 bits,
- un indicateur de retenue (C) de 1 bits,
- un indicateur de zéro (Z) de 1 bit,
- un indication de traitement d'interruption (INTRPT et IFLAG) de 2 bit

La figure III.2.1.b illustre le découpage de l'espace d'adresse. Nous remarquons l'existence de trois types de bus : un *bus adresse* de 15 bits de largeur capable d'adresser une mémoire de 32Ko, un *bus de données* de 8 bits (lié au fait que le cryptoprocesseur traite des données sur 8 bits) et un *bus de contrôle* composé de trois types de signaux qui sont nécessaires pour la sélection du circuit mémoire ainsi que pour la validation des opérations de lecture et d'écriture.

Nous avons donc opté pour la segmentation de la mémoire en trois blocs distincts : PROGRAMME , DONNEES et PILE.

### III.2.3 Structure interne du Cryptoprocesseur

Cette structure est donnée par la figure III.2.2. Nous remarquons l'existence de deux blocs fondamentaux :

- Le chemin de données Cette partie est composée essentiellement de :
  - tous les éléments du modèle de programmation,
  - d'une unité arithmétique et logique.
  - du bloc de chiffrement/déchiffrement
  - d'une unité d'entrée sortie
  - d'un bloc de traitement de l'interruption
- Le séquenceur.

Ces deux parties sont liées à travers le registre d'instruction.

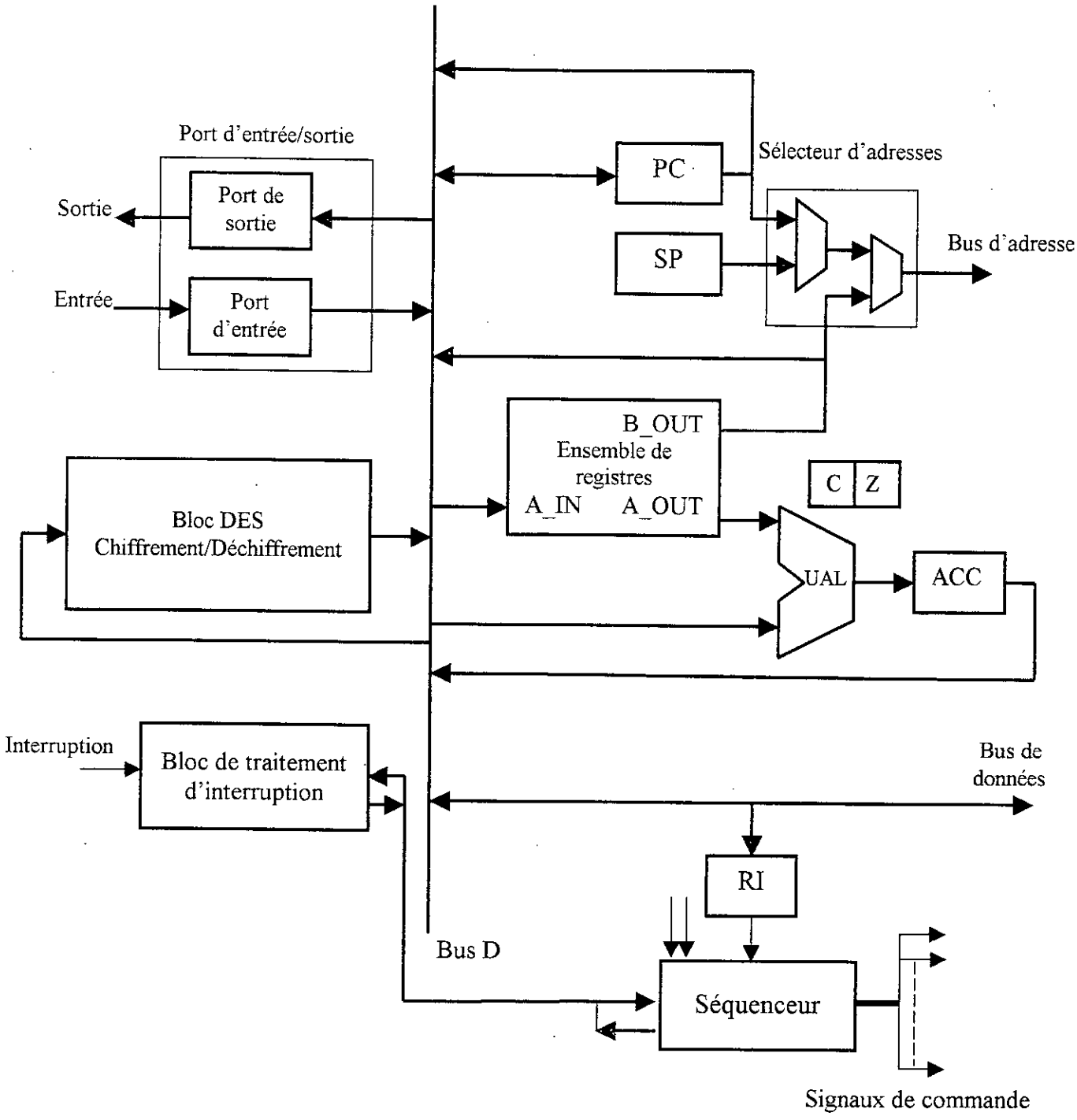



Figure II.2.2 : Structure interne du cryptoprocesseur

### III.2.3.1 Séquenceur

En fonction du code instruction contenu dans le registre d'instruction et du contenu du registre d'état, le séquenceur fourni en sortie un ensemble de signaux de commande qui vont permettre aux autres parties du cryptoprocesseur de réaliser l'opération programmée. Une description détaillée de son fonctionnement est donnée par le paragraphe III.3.2.2.

### III.2.3.2 Bloc chiffrement/déchiffrement

Ce bloc sera dédié à toute opération de chiffrement et de déchiffrement. Il peut être vu comme un bloc autonome doté de son propre chemin de données et de son propre séquenceur réalisant, à la demande, la fonction de chiffrement ciblée. Cette fonction de chiffrement/déchiffrement sera assurée par l'algorithme DES.

## III.2.4 Description comportementale du cryptoprocesseur

Le fonctionnement globale du cryptoprocesseur est illustré par l'organigramme de la figure III.2.3.

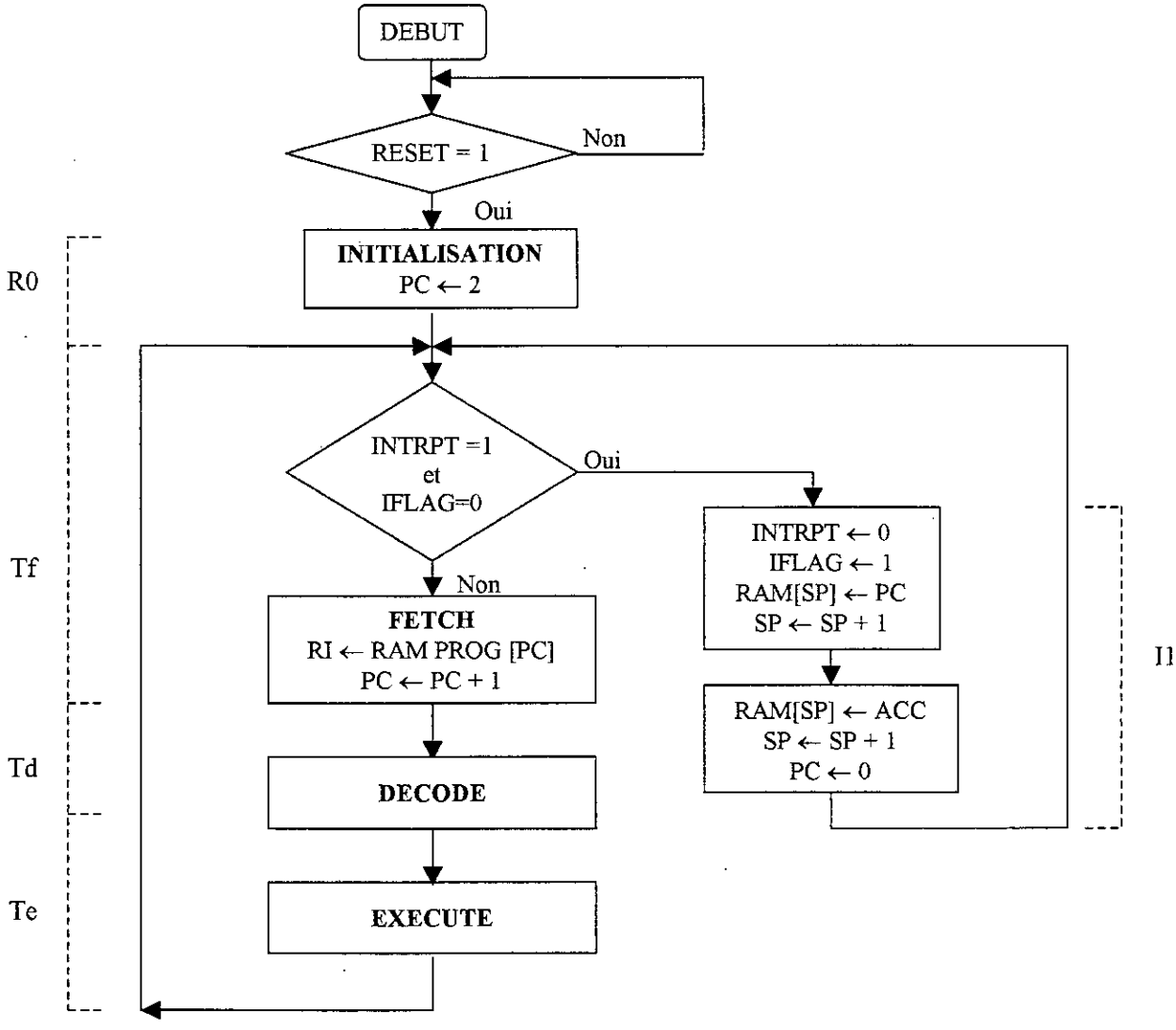



Figure III.2.3 : Organigramme de fonctionnement du cryptoprocesseur

Le fonctionnement global du cryptoprocesseur se résume essentiellement en deux phases. Chaque phase s'exécute sur un certain nombre d'étapes :

- La phase d'initialisation. Après que le cryptoprocesseur ait reçu un signal RESET, celui-ci subit une initialisation global et voit son compteur programme s'incrémenter de deux valeurs.

- La phase de fonctionnement en mode "croisière" où nous distinguons le fonctionnement en mode normal (étapes Tf, Td et Te accessibles pour le traitement des instructions) et le fonctionnement en mode interruptible (étape I1 accessible par le cryptoprocesseur uniquement lorsque une interruption survient).

La description détaillée du fonctionnement du cryptoprocesseur et les conditions de transitions entre ses différentes étapes de fonctionnement sont données ci-dessous.

**ETAPE R0** : Lorsque le cryptoprocesseur reçoit le signal RESET. Il subit une initialisation durant laquelle il y a effacement du contenu des bascules, incrémentation compteur programme de deux valeurs et franchissement à l'étape Tf.

Le fait d'avoir incrémenté le compteur programme de deux adresse implique que tous les programmes du cryptoprocesseur commence à l'adresse 2. La Question qui se pose, quelle est la raison de cette procédure alors que les programmes peuvent débuter à l'adresse zéro ?

La réponse à cette question a une liaison avec le besoin de traiter des interruptions. En effet, l'adresse zéro est réservée au stockage du vecteur d'interruption. Ce dernier possède une instruction de saut au programme prioritaire qui a interrompu le fonctionnement normal du cryptoprocesseur.

**ETAPE Tf** : Pendant cette étape, le cryptoprocesseur vérifie l'état des signaux INTRPT et IFLAG, pour voir s'il y a une demande d'interruption et si un autre programme d'interruption n'est pas en cours d'exécution. Nous pouvons distinguer deux cas :

- Si ces deux conditions sont vérifiées, l'interruption en attente est remise à zéro pour permettre la détection d'une autre interruption, le signal IFLAG est forcé à un pour signaler qu'un programme d'interruption est déjà en cours d'exécution, l'adresse de l'instruction courante est stockée dans la pile, et incrémentation du pointeur de pile. Le cryptoprocesseur rentre ensuite dans un mode de fonctionnement interruptible et ce, en franchissant l'étape I1.

- Sinon, le cryptoprocesseur accède au mode de fonctionnement normal et commence par le cycle de recherche (fetch) des instructions depuis le segment PROGRAMME de la mémoire centrale. Ces instructions seront stockées dans

le registre d'instructions RI, pour être ensuite décoder par le séquenceur dans l'étape suivante Td.

**ETAPE I1** : Ici, le cryptoprocesseur sauvegarde l'accumulateur dans la pile, incrèmente une fois de plus le pointeur de pile et remis à zéro le compteur programme. Il transite à l'étape Tf.

**ETAPE Te** : C'est l'étape d'exécution de l'instruction stockée dans le registre d'instruction (RI). Les tâches effectuées durant cette étape dépendent du code opération de l'instruction à exécuter. L'exécution de certains types d'instructions nécessite jusqu'à trois cycles d'horloge. Donc si l'instruction à besoin d'un cycle supplémentaire, on passe au cycle d'exécution suivant, sinon on transite à l'étape Tf pour le traitement d'une nouvelle instruction.

### III.2.5 Le mode interruptible

Plutôt que d'obliger le cryptoprocesseur à surveiller périodiquement l'état des organes périphériques pour l'échange de données , nous avons jugé préférable d'intégrer la technique des interruptions. A chaque fois qu'un dispositif relié au port d'E/S veut dialoguer avec le cryptoprocesseur, il transmet un signal d'interruption l'obligeant ainsi à interrompre le traitement courant pour permettre au programme réalisant le transfert de s'exécuter en priorité.

Souvent le programme prioritaire utilise les registres internes et le registre d'état du cryptoprocesseur. Il est donc nécessaire pour permettre la reprise du programme interrompu de sauvegarder leur contenu avant le déroutement vers le programme prioritaire. Il est également nécessaire de sauvegarder l'adresse de retour du sous–programme interrompu pour pouvoir le reprendre ultérieurement. Les valeurs du compteur de programme et du registre d'état sont sauvegardées dans la pile alors que les valeurs des registres internes ne sont pas transférées vers la pile. Elles sont toujours sauvegardées dans leurs emplacements initiales des registres internes. Ceci est rendu possible par l'utilisation d'une technique qui permet de découper l'ensemble des registres en deux bancs de huit registres chacun. Chaque banc est réservé à un mode de fonctionnement du cryptoprocesseur (mode normal ou mode interruptible). Un signal fourni par le séquenceur du cryptoprocesseur permet la commutation entre les deux bancs de registres. Cette technique a surtout pour avantage de gagner le temps relatif au transfert des registres internes vers la pile (figure III.2.4).

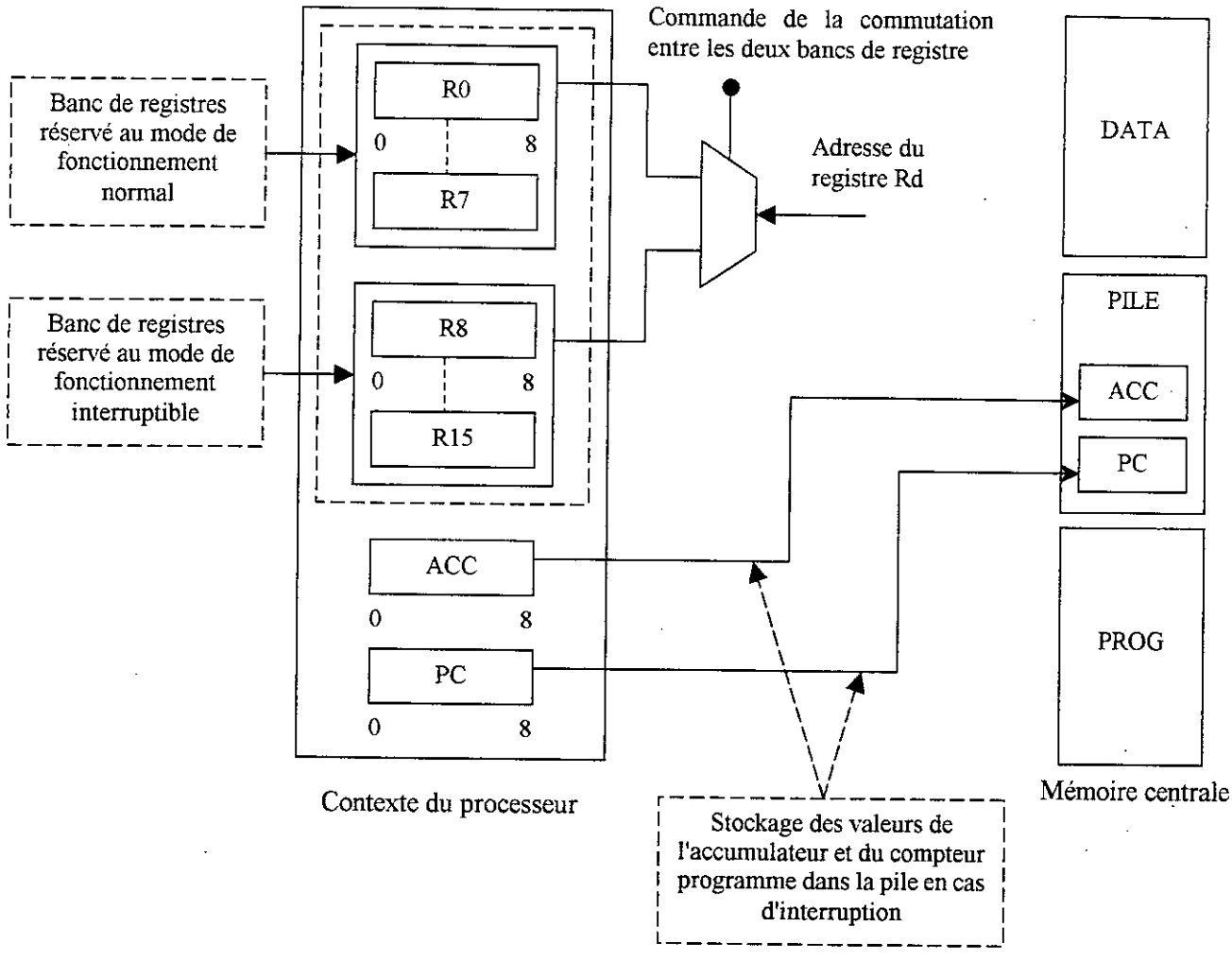



Figure III.2.4 : Sauvegarde du contexte du processeur avant le traitement d'un programme d'interruption

## III.2.6 Cycles fonctionnels et temps d'exécution

Une fois que les instructions sont stockées dans la mémoire programme, le cryptoprocesseur peut lire ces instructions et exécuter ainsi le programme correspondant. Trois cycles élémentaires sont nécessaires pour le traitement de chaque instruction et sont en parfaite adéquation avec le modèle classique de VON NEUMANN:

- **Cycle fetch** : Le cryptoprocesseur doit lire l'instruction courante depuis la mémoire programme et il la stocke dans le registre d'instruction.

- **Cycle decode** : Le cryptoprocesseur doit déduire le type d'instruction qui sera exécutée afin d'obtenir l'opérande correspondant.

- **Cycle execute** : Le cryptoprocesseur doit réaliser les différentes opérations nécessaires sur les opérandes et stocker le résultat correspondant.

La figure III.2.5 illustre le séquencement de ces divers cycles.

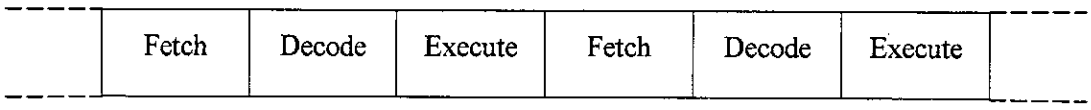



Figure III.2.5 : Cycles fonctionnels du cryptoprocesseur

Le cycle fetch a une durée constante contrairement aux autres cycles dont le temps correspondant dépendrait du mode d'adressage mis en œuvre.

Le temps nécessaire à l'exécution des instructions est noté Te. Il peut varier de un (01) jusqu'à trois (03) cycles d'horloge selon le type de l'instruction à exécuter. Le nombre de cycles nécessaires à l'exécution de chaque instruction est donné par le tableau III.2.1.

## III.2.7 Format des instructions

Le format standard d'une instruction est donné par la figure III.2.6. C'est un modèle d'instruction à un seul opérande. L'instruction est composée de trois champs. Le premier champ est réservé pour le code instruction afin de distinguer entre les différents types d'instructions ; le second sert au stockage de l'adresse du registre interne (Rd, d variant de 0 à 16) utilisé par ces instructions et le troisième champ est réservé au stockage de l'opérande.

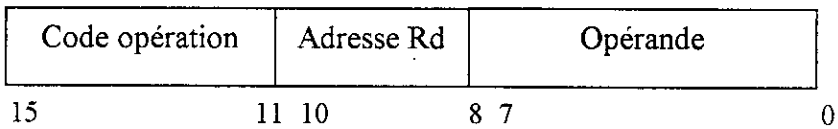



Figure III.2.6 : Format standard d'une instruction

## III.2.8 Modes d'adressage [34, 35]

Quatre modes d'adressage sont considérés pour les différentes raisons suivantes :

- *L'adressage immédiat*, dans certains cas nous avons besoin de charger immédiatement le contenu de l'accumulateur avec une valeur stockée dans le code opérande au lieu d'aller là chercher depuis la mémoire. Ces valeurs peuvent être l'adresse de début d'une liste de nombres, à traiter, stockés dans le segment DONNES de la mémoire centrale.

- *L'adressage direct*, très souvent nous avons besoin de transférer des valeurs de la mémoire DONNEES vers l'accumulateur et vis versa. Pour cela un adressage direct par registre est utilisé.

- *adressage indexé*, le besoin exprimé en terme de traitement de blocs de données nous conduit à l'implémentation de ce mode d'adressage.

- *adressage indirect*, similaire au mode d'adressage indexé sauf que dans certains cas nous avons besoin de traiter des blocs de données tout en stockant le registre d'index dans la mémoire centrale.

### III.2.9 Jeu d'instructions

Le choix du jeu d'instructions dépend de l'approche d'exploitation des cycles fonctionnels ainsi que des optimisations éventuelles à cibler. Ce qui nous conduit à faire un choix entre les deux approches CISC et RISC. La deuxième dans notre cas a été écartée en raison de l'objectif ciblé d'une part et du choix du type de séquencement pour les cycles de fonctionnement d'autre part. C'est ainsi que l'approche CISC s'est imposée.

Le tableau III.2.1 donne le jeu d'instructions supporté par le cryptoprocesseur où nous pourrons distinguer deux classes d'instructions : les instructions à usage général et les instructions spécifiques aux opérations de chiffrement et de déchiffrement.

#### a) Instructions à usage général

Les instructions à usage général supportées par le cryptoprocesseur sont les suivantes :

- Les instructions de transfert (load et store). Il s'agit essentiellement de transferts de mots entre accumulateur et registres ou entre accumulateur et mémoire.

- Les instructions arithmétiques. Il s'agit de deux opérations ADD et XOR.

- Les instructions de manipulations des registres d'états (clear_c, set_c et test).
- Les instructions de branchements ( jump #adresse, jc et jz). Ces instructions permettent d'exécuter des ruptures de séquence soit inconditionnels, soit dépendant de conditions portant sur les indicateurs d'états.

- Les instructions d'appel et de retour de procédure (jsr et ret). Permettent d'appeler des sous-programme, tout en conservant l'adresse de retour au programme principale dans la pile.

- Les instruction (reti), de restauration de l'état du processeur, tel qu'il a été avant l'exécution du programma d'interruption. Ainsi que la reprise de l'exécution des instructions à l'adresse indiquée par le PC.

- Les instructions d'E/S à usage général. Elles servent au stockage, dans les registres internes du processeur, des données qui arrivent au niveau du port d'entrée, et à l'envoi du contenu de l'accumulateur vers le port de sortie.

**b) Instructions de Chiffrement/Déchiffrement**

Elles sont spécifiques aux opérations de chiffrement/déchiffrement. Nous trouvons dans cette classe :

- Les instructions de chiffrement (encm et decp). Ces instructions sont spécifiques aux opérations de chiffrement de la mémoire et de déchiffrement du port de sortie.

- Les instructions d'initialisation et de test de l'état des tampons du bloc fonctionnel de chiffrement/déchiffrement (initbuf, jbif et jbof). Ces instructions permettent d'initialiser, au début d'une opération de chiffrement ou de déchiffrement, les tampons ainsi que le teste de leurs états.

- Les instructions d'E/S propre aux opérations de chiffrement/déchiffrement (out_bo). Elles sont nécessaires au transfert des données entre le bloc fonctionnel de chiffrement/déchiffrement et le port d'E/S.

| Mnémonique | Opérations | Descriptions | Temps d'exécution Te (nbre de cycles) |
|---|---|---|---|
| load Rd | ACC ← Rd | Charger l'accumulateur avec le contenu du registre Rd. d est une adresse dans l'intervalle [0..7]. | 1 |
| load (Rd) | ACC ← RAM[Rd] | Charger l'accumulateur avec le contenu de la case mémoire DONNEES, dont l'adresse se trouve dans le registre Rd. | 1 |
| load #d | ACC ← d | Charger l'accumulateur avec la valeur d. d est dans l'intervalle [0..255]. | 1 |
| load d | ACC ← RAM[d] | Charger l'accumulateur avec le contenu de la case mémoire DONNEES, d'adresse d. d est dans l'intervalle [0..255]. | 2 |
| load (d) | ACC ← RAM[RAM[d]] | Charger l'accumulateur avec le contenu de la case mémoire (RAM DATA), dont l'adresse se trouve dans la case mémoire DONNEES d'adresse d. d est dans l'intervalle [0..255]. | 3 |
| store Rd | Rd ← ACC | Stocker l'accumulateur dans le registre Rd. d est dans l'intervalle [0..7]. | 1 |
| store (Rd) | RAM[Rd] ← ACC | Stocker l'accumulateur dans la mémoire DONNEES à l'adresse trouvées dans le registre Rd. d est une adresse dans l'intervalle [0..7]. | 1 |
| store d | RAM[d] ← ACC | Stocker l'accumulateur dans la mémoire DONNEES à l'adresse d. d est dans l'intervalle [0..255]. | 2 |
| store (d) | RAM[RAM[Rd]] ← ACC | Stocker l'accumulateur dans la case mémoire DONNEES, dont l'adresse se trouve dans la mémoire DONNEES d'adresse se trouvant dans le registre Rd. d est dans l'intervalle [0..7]. | 3 |
| in Rd | Rd ← IN_PORT | Charger le registre Rd avec la valeur du port d'entrée. | 1 |
| out | OUT_PORT ← ACC | Vider le contenu de l'accumulateur dans le port de sortie. | 1 |
| xor Rd | ACC ← ACC$Rd | Opération xor du registre Rd avec l'accumulateur. d est une adresse dans l'intervalle [0..7]. | 1 |

Tableau III.2.1 : Jeu d'instructions et temps relatif d'exécution

| Mnémonique | Opérations | Descriptions | Temps d'exécution Te (nbre de cycles) |
|---|---|---|---|
| add Rd | ACC ← ACC + Rd + C ;<br>C ← carry out | Additionner le contenu du registre Rd et l'indicateur de retenue C avec l'accumulateur. d est une adresse dans l'intervalle [0..7]. | 1 |
| test Rd | Z ← ACC & Rd | Opération AND entre le contenu du registre Rd et l'accumulateur, mais au lieu de stocker le résultat dans l'accumulateur, mettre à 1 le registre d'état Z si le résultat est 0 ; sinon mettre à 0 le registre d'état Z. d est une adresse dans l'intervalle [0..7]. | 1 |
| clear_c | C ← 0 | Effacer l'indicateur de retenue C. | 1 |
| set_c | C ← 1 | Mettre à 1 l'indicateur de retenue C. | 1 |
| jc #a | IF C = 1 → PC ← a<br>Else PC ← PC + 2 | Si C = 1, brancher à l'adresse a. si C = 0, continuer à la prochaine instruction. a est dans l'intervalle [0..255]. | 1 |
| jz #a | IF Z = 1 → PC ← a<br>Else PC ← PC + 2 | Si Z = 1, brancher à l'adresse a. si Z = 0, continuer à la prochaine instruction. a est dans l'intervalle [0..255]. | 1 |
| jump #a | PC ← a | Branchement inconditionnel à l'adresse a. a est dans l'intervalle [0..255]. | 1 |
| jsr #a | RAM[SP] ← PC + 2<br>SP ← SP +1<br>PC ← a | Sauvegarder l'adresse de la prochaine instruction dans la PILE et incrémenter le pointeur de pile. Ensuite sauter à l'adresse a. a est dans l'intervalle [0..255]. | 3 |
| Ret | SP ← SP – 1<br>PC ← RAM[SP] | Décrémenter le pointeur de pile et charger le PC avec l'adresse trouvée dans la PILE. | 2 |
| Reti | SP ← SP – 1<br>ACC ← RAM[SP]<br>SP ← SP – 1<br>PC ← RAM[SP] | Décrémenter le pointeur de pile et charger l'accumulateur avec la valeur trouvée dans la PILE. Ensuite décrémenter SP et charger le PC avec l'adresse de reprise du programme de non interruption. | 3 |
| encm (Rd) | BI ← RAM [Rd] | Charger l'accumulateur avec le contenu de la case mémoire DONNES, dont l'adresse se trouve dans le registre Rd. | 1 |

Tableau III.2.1 : Jeu d'instructions et temps relatif d'exécution (suite)

| Mnémonique | Opérations | Descriptions | Temps d'exécution Te (nbre de cycles) |
|---|---|---|---|
| initbuf | RADI ← 1 | Chargement du registre à décalage par un logique. | 1 |
| jbif | IF BIF = 1 → PC ← a<br>Else PC ← PC + 2 | Si BIF = 1, brancher à l'adresse a. Si BIF = 0, continuer à la prochaine instruction. a est dans l'intervalle [0..255]. | 1 |
| jbof | IF BOF = 1 → PC ← a<br>Else PC ← PC + 2 | Si BOF = 1, brancher à l'adresse a. Si BOF = 0, continuer à la prochaine instruction. a est dans l'intervalle [0..255]. | 1 |
| out_bo | OUT_PORT ← BO | Vider le contenu du tampon de sortie dans le port de sortie | 1 |
| Decp | BI ← IN_PORT | Charger le contenu du port d'entrée dans le tampon d'entrée | 1 |

Tableau III.2.1 : Jeu d'instructions et temps relatif d'exécution (suite)

### III.2.10 Encodage des instructions

L'encodage du jeu d'instructions est donné par tableau III.2.2. Les trois derniers bits de poids faible de l'instruction servent au stockage de l'adresse du registre interne utilisé par l'instruction. Ces trois bits sont mis à zéro dans le cas des instructions ne manipulant pas des registres internes. Les cinq bits restant sont utilisés pour distinguer entre les différents types d'instructions. Nous mentionnons l'existence d'instructions codées sur 2 octets.

| Mnémonique | # Octets | Encodage |
|---|---|---|
| load Rd | 1 | 00100 d2 d1 d0 |
| load (Rd) | 1 | 00101 d2 d1 d0 |
| load #d | 2 | 01000000 d7 d6 d5 d4 d3 d2 d1 d0 |
| load d | 2 | 00110000 d7 d6 d5 d4 d3 d2 d1 d0 |
| load (d) | 2 | 00111000 d7 d6 d5 d4 d3 d2 d1 d0 |
| store Rd | 1 | 00000 d2 d1 d0 |
| store (Rd) | 1 | 00001 d2 d1 d0 |
| store d | 2 | 00010000 d7 d6 d5 d4 d3 d2 d1 d0 |
| store (d) | 2 | 00011000 d7 d6 d5 d4 d3 d2 d1 d0 |
| in Rd | 1 | 01100 d2 d1 d0 |
| out | 1 | 01101000 |
| xor Rd | 1 | 10000 d2 d1 d0 |
| add Rd | 1 | 10001 d2 d1 d0 |
| test Rd | 1 | 10010 d2 d1 d0 |
| clear_c | 1 | 10100000 |
| set_c | 1 | 10101000 |
| jc #a | 2 | 11000000 a7 a6 a5 a4 a3 a2 a1 a0 |
| jz #a | 2 | 11001000 a7 a6 a5 a4 a3 a2 a1 a0 |
| jump #a | 2 | 11010000 a7 a6 a5 a4 a3 a2 a1 a0 |
| jsr #a | 2 | 11011000 a7 a6 a5 a4 a3 a2 a1 a0 |
| ret | 1 | 11100000 |
| reti | 1 | 11101000 |
| encm | 1 | 11110 d2 d1 d0 |
| initbuf | 1 | 01010000 |
| Jbif | 1 | 01011000 |
| Jbof | 1 | 01110000 |
| out_bo | 1 | 01111000 |
| Decp | 1 | 10111000 |

Tableau III.2.2 : Encodage des instructions

## III.2.11 Port d'entrée/sortie (E/S)

Les échanges d'information entre le cryptoprocesseur et ses organes périphériques se fait grâce au port d'E/S. Huit lignes d'E/S jouent le rôle d'une interface entre le chemin de données du cryptoprocesseur et l'environnement extérieur (voir figure III.2.7). Le cryptoprocesseur aura donc besoin de signaux de contrôle appropriés de telle sorte qu'il n'y ait pas de collision entre les données du port d'E/S et les données lues à partir de la RAM. Le port d'E/S peut être adressé comme s'il était un des registres internes du cryptoprocesseur. Ceci à pour avantage de permettre au cryptoprocesseur de dialoguer avec le port d'E/S en utilisant toutes les ressources que peuvent accéder les registres internes.

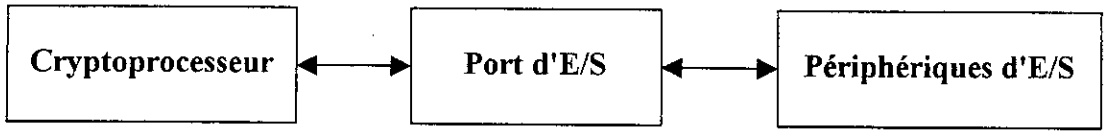



Figure III.2.7 : Le port d'entrée/sortie

## III.2.12 Conclusion

Le modèle comportemental dégagé pour l'implémentation du cryptoprocesseur déduit donc un modèle d'exécution séquentiel à des temps d'exécution hétérogènes. Ce qui conduit implicitement à un modèle à classification SISD de type CISC.

Le paragraphe suivant présentera le modèle structurel du cryptoprocesseur au travers de l'étude détaillée des deux parties qui le constituent, à savoir, le chemin de données et le séquenceur. Il fera essentiellement l'objet d'une étude concernant la manière d'interfacer le bloc de chiffrement/déchiffrement avec les autres parties du cryptoprocesseur et les signaux de commande à prévoir à cet effet.

# III.3 - Modèle structurel du cryptoprocesseur

## III.3.1 Introduction

Dans ce chapitre nous présentons la description globale de architecture du cryptoprocesseur, ensuite nous détaillerons par la suite les deux parties qui la constituent, à savoir, chemin de données et séquenceur.

## III.3.2 Description de l'architecture globale

Dans le but de réaliser le cryptoprocesseur, nous avons procédé à la traduction architecturale de l'organigramme de fonctionnement du cryptoprocesseur par la conception de deux unités principales :

- chemin de données.
- séquenceur

### III.3.2.1 Conception fonctionnelle du Chemin de données

Le chemin de données (figure III.3.1) et l'ensemble des organes permettant de transférer, mémoriser ou traiter les informations (instructions, adresses, opérandes) issues de la mémoire externe ou des organes périphériques. Les composants du chemin des données répondent à l'une de ces trois fonctions :

- Les registres et les bascules pour le stockage d'informations,
- Les unités fonctionnelles pour le traitement d'informations,
- Les bus d'interconnexion pour les transferts d'informations,

**a) Les registres et les bascules**

Nous trouvons :

- un registre d'instruction (RI) de 8 bits, utilisé pour le stockage de l'instruction qui est en cours d'exécution,
- l'accumulateur (ACC) de 8 bits,
- le compteur programme (PC) de 8 bits, utilisé pour le stockage de l'adresse de l'instruction qui est en cours d'exécution,
- le pointeur de pile (SP) de 8 bits, contient l'adresse de la pile, où vont être stockés le compteur programme et l'accumulateur pendant l'exécution des instructions d'appel à un sous–programme et de traitement d'interruptions,

- un sélecteur d'adresse mémoire. Le cryptoprocesseur divise la mémoire externe en trois segment : PROGRAMME (256 octets), DATA (256 octets) et PILE (256 octets). Le sélecteur d'adresse valide l'un des trois segments et lui envoi une adresse provenant de l'une des trois sources : le segment PROGRAMME est adressé par le compteur programme, le segment PILE est adressé par le pointeur de pile et le segment DATA est adressé par la valeur contenu dans l'un des registres internes,

- l'indicateur de zéro Z de 1 bit,

- l'indicateur de retenue C de 1 bit,

- l'ensemble de registres, le cryptoprocesseur possède 16 registres internes. Huit d'entre eux sont disponibles pour un usage normal, et les huit autres sont utilisés pendants le traitement de l'interruption. Cet ensemble de registres est conçu à base de mémoire à double port avec deux sorties à 8 bits (A_OUT et B_OUT) et une seule entrée (A_IN) à 8 bits. Ceci permet à l'ensemble de registres de faire sortir deux valeurs et de réécrire une d'entre elles en un seul cycle d'horloge.

- deux bascules appartenants au bloc de traitement d'interruption, la première bascule sert à maintenir en attente les interruptions qui surviennent, la seconde indique au cryptoprocesseur qu'un programme d'interruption est en cours d'exécution.

**b) Les unités fonctionnelles**

Existence de deux unités fonctionnelles :

- une unité arithmétique et logique (UAL) de 8 bits, effectue les opérations ADD et XOR sur les deux opérandes en entrée et stocke le résultat dans l'accumulateur (ACC). Elle se sert des registres d'états C et Z, respectivement, pour le stockage de la retenue et l'indication d'un résultat de calcul nul,

- un bloc fonctionnel de chiffrement/déchiffrement, assure les opérations de chiffrement/déchiffrement. Il possède une entrée et une sortie de 8 bits chacune reliées au bus interne du processeur (bus D).

**c) Les bus d'interconnexions**

Un bus D, relie la plupart des composants du cryptoprocesseur entre eux. Il peut être chargé par le compteur programme (PC), la sortie B de l'ensemble de registres, l'accumulateur, le port d'entrée et le bus de données externe qui connecte la RAM au cryptoprocesseur. Dans l'autre sens, les valeurs sur le bus D peuvent être chargées dans PC, l'entrée A_IN de l'ensemble de registres, l'accumulateur, le port de sortie et la RAM.

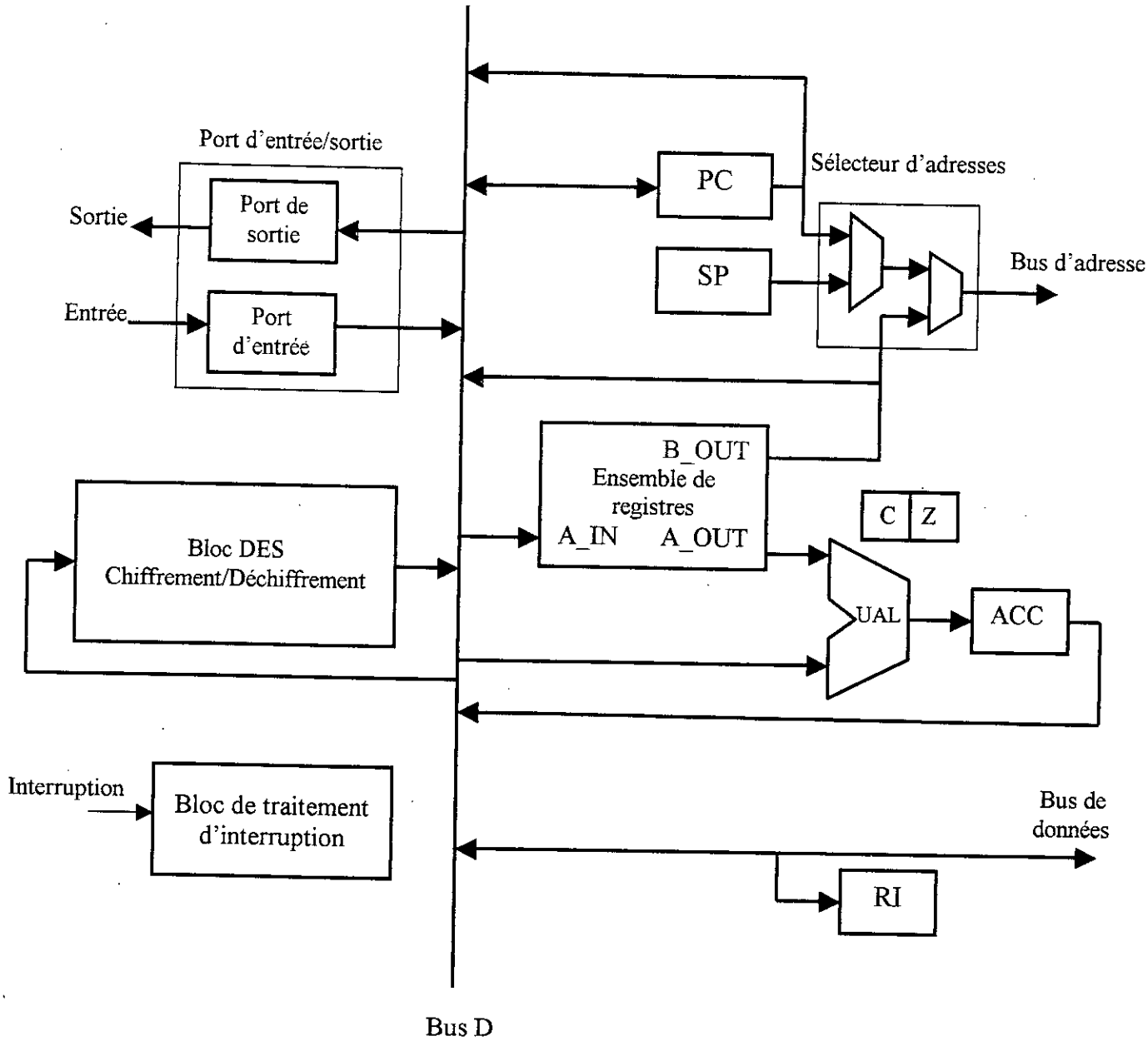



Figure III.3.1 : Chemin de données

### III.3.2.2 Signaux de commande du chemin des données

Tout au long du chemin des données sont distribués des signaux de commande permettant l'exécution des différentes opérations de transfert, stockage et traitement d'informations. Ces signaux de commande sont issus du séquenceur.

La figure III.3.2 montre les signaux de commande du chemin des données fournis par le séquenceur. Nous trouvons les signaux de :

- chargement de registres,

- lecture des registres sur le bus D,
- remise à zéro des registres,
- commande de l'UAL,
- commande du bloc DES
- commande des indicateurs, de zéro et de retenue,
- commande de la mémoire centrale,
- commande du bloc de traitement d'interruption.

### III.3.2.3 Identification des signaux de commande

Le tableau III.3.1 donne la description de chaque signal de commande.

| Signaux | Descriptions |
|---|---|
| LD_PC | Chargement du PC avec la valeur présente sur le bus D |
| INC_PC | Incrémentation de la valeur du PC |
| CLR_PC | Effacement du contenu du PC |
| DRV_D_PC | Ecriture du contenu du PC sur le bus D |
| CHG_SP | Changement du contenu du SP |
| INC_SP | Sa valeur provoque dans SP, soit une opération d'incrémentation ou de decrémentation |
| ADDR_SEL[1:0] | Sélection du segment mémoire à adresser (programme, données ou pile) |
| LD_REG | Chargement d'un registre interne par la valeur présente sur le bus D |

Tableau III.3.1 : Signaux de commande du séquenceur

| Signaux | Descriptions |
|---|---|
| R0A | Adressage du registre interne R0 appartenant au banc de registres du mode normal |
| R0B | Adressage du registre interne R0 appartenant au banc de registres du mode interruptible |
| IFLAG | Sélection d'un banc de huit registres de l'ensemble de registre interne |
| DRV_D_REG | Ecriture de la sortie B_OUT de l'ensemble de registres sur le bus D |
| S0, S1 et ADD | Contrôle des différentes opérations qui s'exécutent dans l'UAL |
| LD_ACC | Chargement de l'accumulateur |
| LD_CRY | Chargement de l'indicateur de retenue |
| ALU_CRY | Chargement de l'indicateur de retenue avec la sortie retenue de l'additionneur 8 bits |
| SET_CRY | Mise à 1 ou à 0 de l'indicateur de retenue |
| LD_Z | Lecture de l'indicateur Z |
| DRV_D_ACC | Ecriture du contenu de l'accumulateur sur le bus D |
| WRITE | Ecriture du contenu du bus D sur le bus de données de la mémoire centrale |
| DRV_D_DATA | Ecriture du contenu du bus de données de la mémoire centrale sur le bus D |
| LD_IR | Chargement du registre d'instruction |
| LD_PORT | Chargement du port de sortie |
| DRV_D_PORT | Ecriture du contenu du port d'entrée sur le bus D |
| LD_BI | Chargement du tampon d'entrée du module DES |
| DRV_BO | Lecture du contenu du tampon de sortie du module DES |
| ED | Sélection entre une opération de chiffrement et une opération de déchiffrement |
| IE | Activation du tampon d'entrée du module DES |
| DRV_D_DES | Ecriture du contenu de module DES sur le bus D |
| SET_IFLAG | Mise à 1 du signal (IFLAG) indiquant l'exécution d'un programme d'interruption |
| CLR_IFLAG | Mise à 0 du signal IFLAG |
| CLR_INTRPT | Remise à 0 du circuit de maintient des interruptions |

Tableau III.3.1 : Signaux de commande du séquenceur (suite)

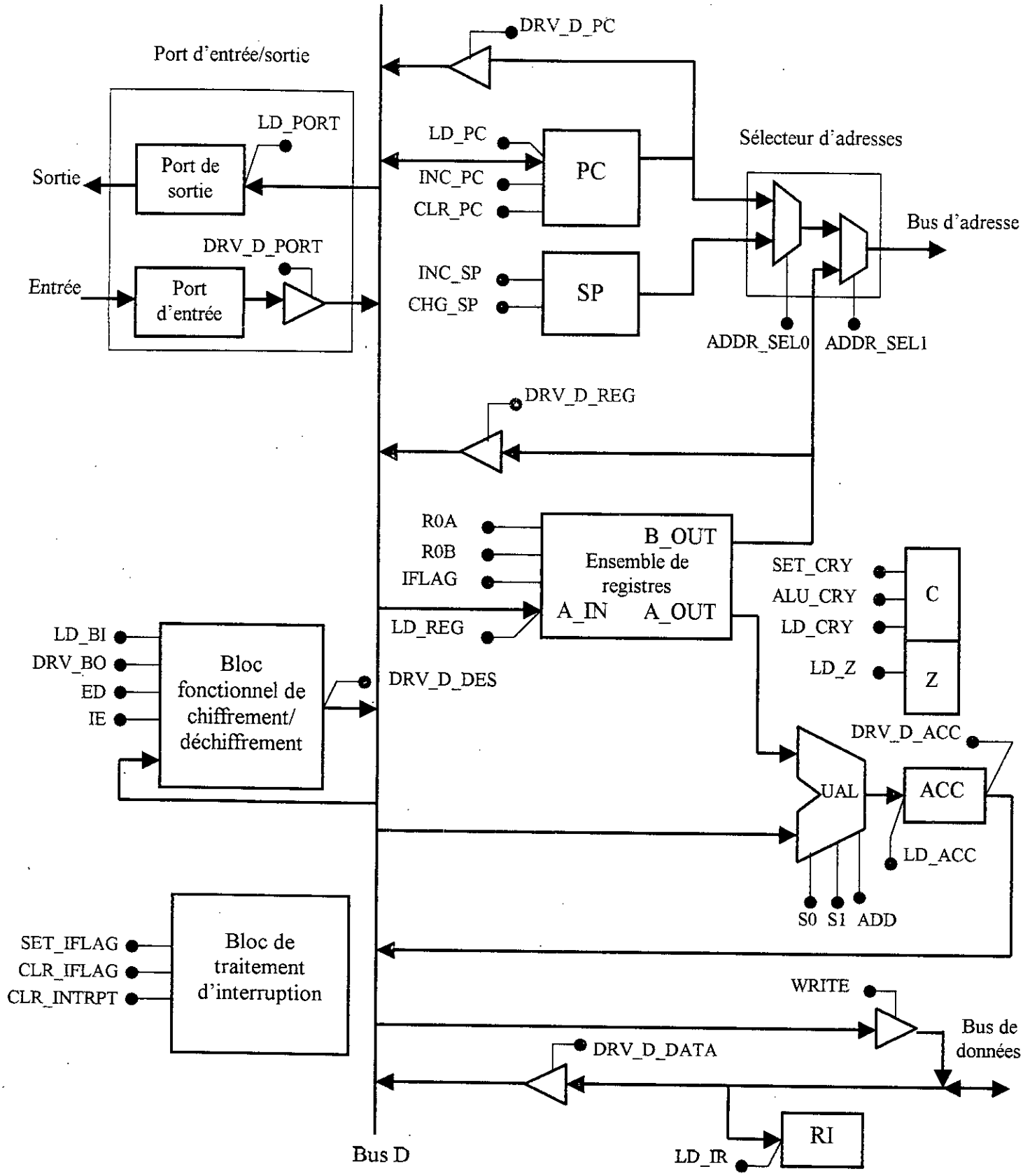
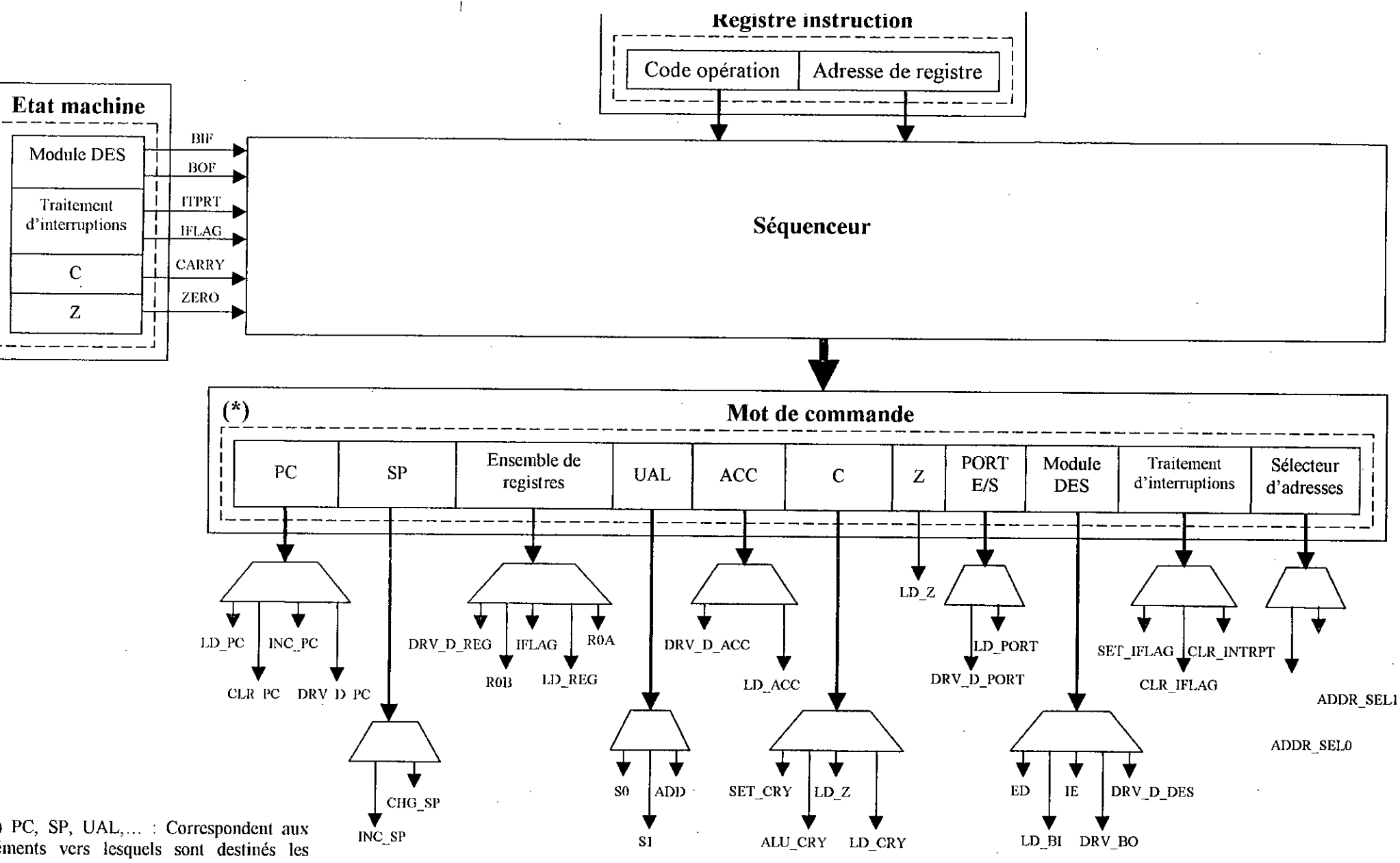



Figure III.3.2 : Signaux de commande et chemin de données

Nous pouvons déjà définir le séquenceur de façon externe, c'est à dire par ses informations d'entrées et de sorties (figure III.3.3) :

- Les informations de sortie du séquenceur ne sont autres que les signaux de commande qui doivent être distribués le long du chemin de données,

- Les informations d'entrée du séquenceur sont fournies d'une part par l'instruction, ce sont le code opération et les adresses des registres internes, et d'autre part par l'état machine qui regroupe un certain nombre d'informations telles que l'état des indicateurs de retenue et de zéro, les demandes d'interruptions, les indicateurs d'état des unités fonctionnelles.




) PC, SP, UAL,... : Correspondent aux éments vers lesquels sont destinés les gnaux de commande

Figure III.3.3 : Mot de commande, signaux d'états et signaux de commande

### III.3.3 Fonctionnement détaillé du chemin de données

Ce paragraphe est consacré au le fonctionnement détaillé des trois parties constituant le chemin de données : les unités fonctionnelles, les registres et les bascules, et le bus d'interconnexion

#### III.3.3.1 Les unités fonctionnelles

#### a) L'unité arithmétique et logique

L'unité arithmétique et logique (UAL) est composé de plusieurs opérateurs spécialisés chacun dans l'exécution d'un ou plusieurs types d'opérations invoquées dans le jeu d'instructions du cryptoprocesseur.
L'UAL (figure III.3.4) contient les composants logiques nécessaires pour l'exécution des opérations d'addition, et de OU exclusif, pour le stockage de l'accumulateur et des indicateurs de retenue et de zéro. Les deux opérandes de 8 bits entrent dans l'UAL à travers les entrées [R7..R0] et [A7..A0] provenants respectivement de la sortie A_OUT de l'ensemble de registres et du Bus D.
L'exécution des différentes opérations dans l'UAL est contrôlée par les signaux de commandes ADD, S0 et S1. La table de vérité de l'UAL est présentée par le tableauIII.3.2.

L'indicateur de zéro est toujours chargé avec la sortie de la porte NOR. Les entrées de la porte NOR sont le résultat d'une opération AND entre les valeurs issues du bus D et de l'ensemble de registres (la figure III.3.5 montre les détails du circuit de l'indicateur de zéro (ZEROTEST)).

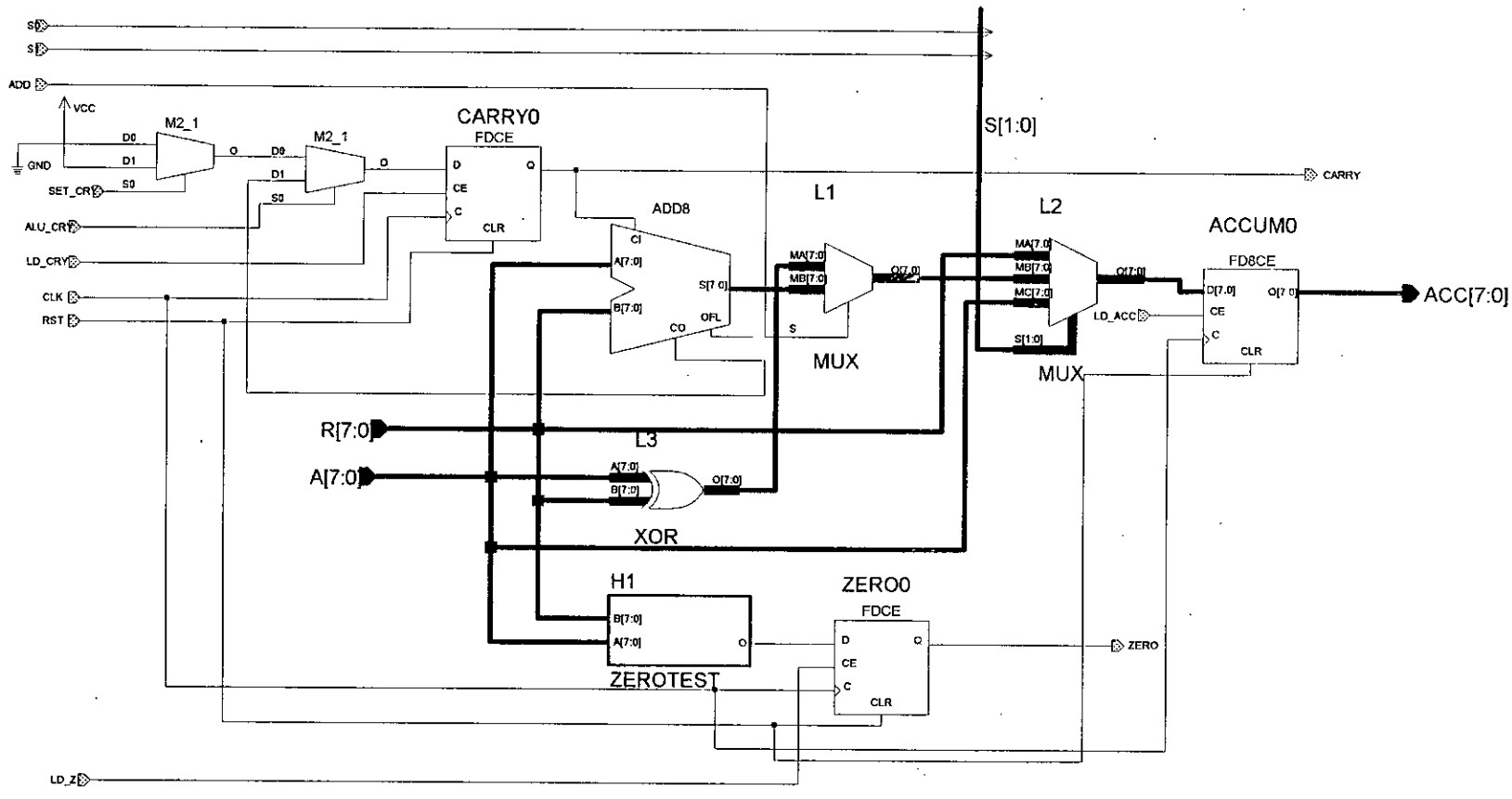



Figure III.3.4 : Circuit de l'UAL

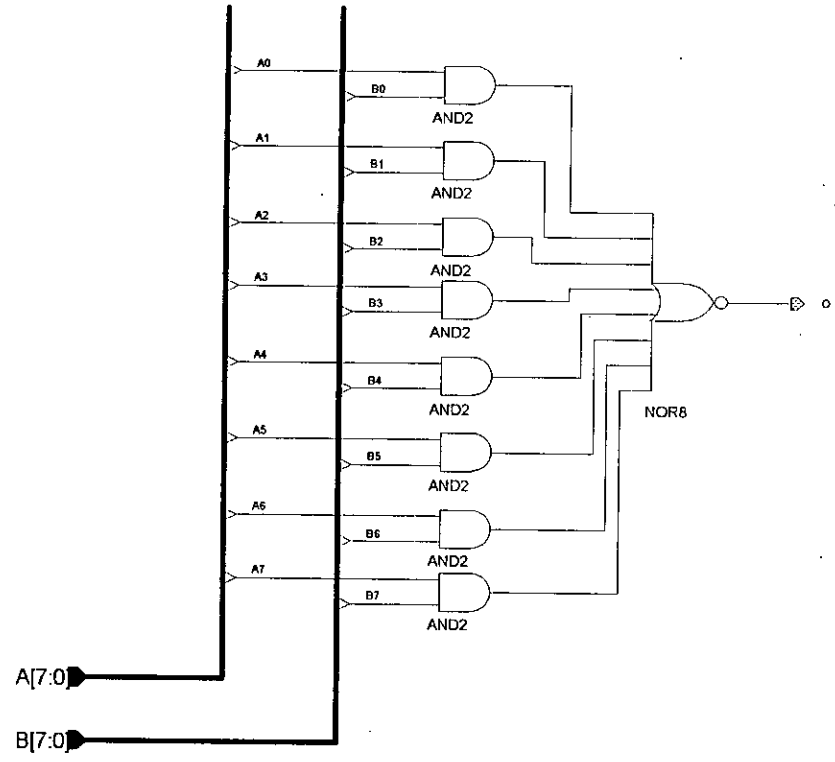



Figure III.3.5 : Circuit de l'indicateur de zéro

| INPUT | | | | | | | | | | OUTPUT | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ADD** | **S0** | **S1** | **LD_ACC** | **LD_Z** | **LD_CRY** | **ALU_CRY** | **SET_CRY** | **RESET** | **CLK** | **ZERO** | **CARRY** | **ACC** |
| 0 | 1 | 0 | 1 | | | | | 0 | ↑ | | | ACC ← A[7 :0]$R[7 :0] |
| 1 | 1 | 0 | 1 | | | | | 0 | ↑ | | | ACC ← A[7 :0] + R[7 :0] + CARRY |
| X | 0 | 1 | 1 | | | | | 0 | ↑ | | | ACC ← A[7 :0] |
| X | 0 | 0 | 1 | | | | | 0 | ↑ | | | ACC ← R[7 :0] |
| | | | | 1 | | | | 0 | ↑ | (ACC & Rd) | | |
| | | | | | | | | 0 | ↑ | | | |
| | | | | | 1 | 1 | X | 0 | ↑ | | CO | |
| | | | | | 1 | 0 | 0 | 0 | ↑ | | 0 | |
| | | | | | 1 | 0 | 1 | 0 | ↑ | | 1 | |
| | | | | | | | | 1 | ↑ | 0 | 0 | 00000000 |

Tableau III.3.2 : Table de vérité de l'UAL

### b) Le bloc fonctionnel Chiffrement/Déchiffrement

Le bloc fonctionnel chiffrement/déchiffrement repose sur l'algorithme cryptographique DES. Ce bloc fonctionnel n'est rien d'autre que notre conception à base d'une architecture standard de l'algorithme DES (voir paragraphe II.2.2). Chacune des options architecturales (en boucle déroulée, en pipeline ou combiné) déjà implémentées sous les différentes versions citées dans le tableau II.2.1 peuvent être utilisées comme bloc fonctionnel de chiffrement et de déchiffrement.

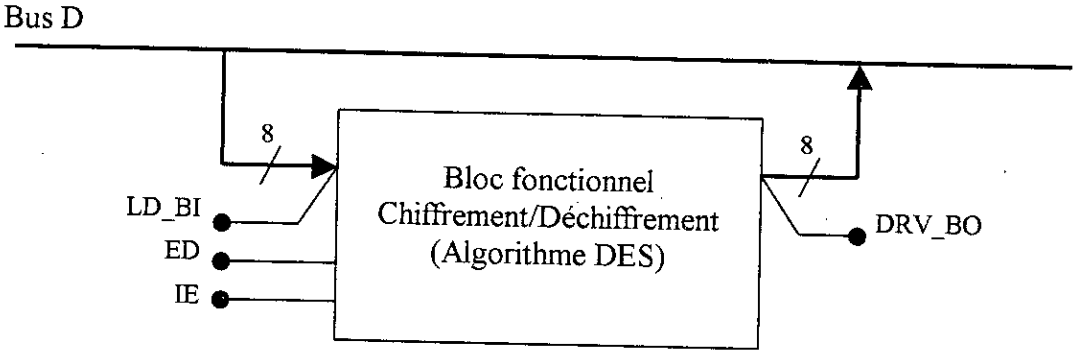



Figure III.3.6 : Vue d'ensemble du bloc fonctionnel chiffrement/déchiffrement

Tel que illustré par la figure III.3.6 le bloc fonctionnel de chiffrement et de déchiffrement est relié à l'ensemble des ressources du cryptoprocesseur par l'intermédiaire du bus D. Les opérations d'écriture et de lecture sur ce bus d'interconnexion se font grâce aux signaux de commande respectifs LD_BI et DRV_BO fournis par le séquenceur du cryptoprocesseur. D'autres signaux de commande sont nécessaires à la gestion du fonctionnement de ce bloc, dont leurs rôles seront décrits dans le tableau III.3.3.

L'algorithme DES chiffre des blocs de données de 64 bits (voir sous-chapitre I.4). Mais il reçoit ces données à travers le bus D sous forme de blocs de 8 bits, ceci nous amène à prévoir des tampons en entrées et en sortie du circuit DES et prévoir aussi les signaux de commande et d'état qui leurs sont associés (voir tableau III.3.3). La figureIII.3.7 donne une vue détaillée du bloc fonctionnel de chiffrement/déchiffrement.

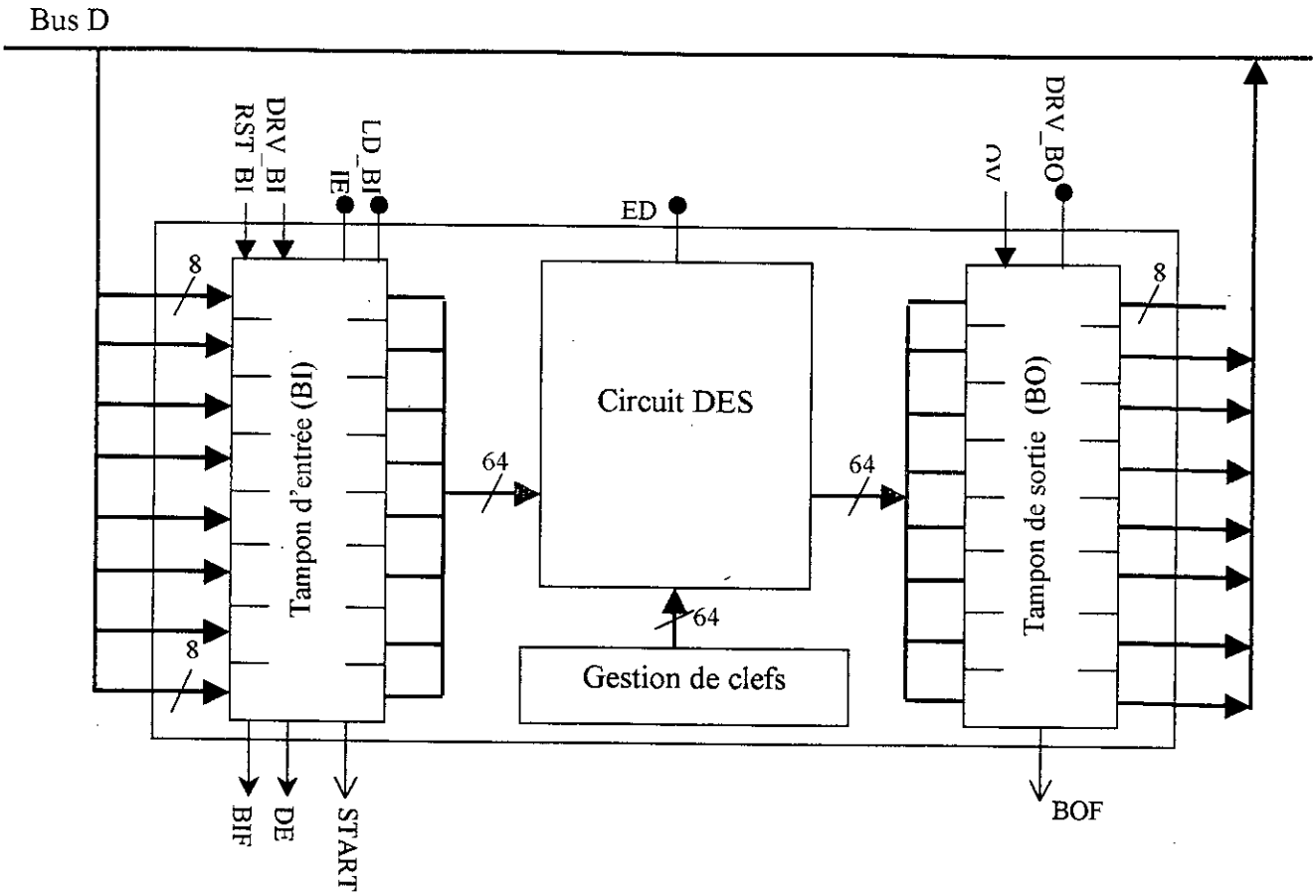



Figure III.3.7 : Schéma détaillé du bloc fonctionnel de chiffrement/déchiffrement

**Légendes**

Commande du bloc fonctionnel chiffrement/Déchiffrement (par le séquenceur du cryptoprocesseur)

signaux d'état (vers le séquenceur du cryptoprocesseur)

commande des tampons d'entrée/sortie (par le séquenceur du circuit DES)

signaux d'état (vers le séquenceur du circuit DES)

**Identification des signaux**

L'ensemble des signaux de commande et d'état qui sont associés au bloc fonctionnel de chiffrement/déchiffrement seront décrits dans la tableau ci-dessous.

| Signaux | Description |
|---|---|
| IE | Activation du tampon d'entrée |
| LD_BI | Chargement du tampon d'entrée par les données présentes sur le bus D |
| DRV_BI | Vidage du tampon d'entrée dans le circuit DES du contenu |
| RST_BI | Désactiver les données contenues dans le tampon d'entrée après les avoir traiter par le circuit DES |
| START | Déclenchement de l'opération chiffrement/déchiffrement |
| DE | Données valides, les données contenues dans le tampon d'entrée sont valides et peuvent être traitées par le circuit DES |
| BIF | Indique au cryptoprocesseur que le tampon d'entrée est plein |
| OV | Sortie valide, signal de chargement dans le tampon de sortie des données traitées par le circuit DES |
| DRV_BO | Lecture du contenu du tampon de sortie |
| BOF | Indique au cryptoprocesseur que le tampon de sortie est plein |
| DRV_D_DES | Ecriture du contenu du bloc fonctionnel de chiffrement/déchiffrement sur le bus D |
| ED | Sélection entre une opération de chiffrement et une opération de déchiffrement |

.Tableau III.3.3 : Signaux de commande et signaux d'état associés au bloc fonctionnel de chiffrement/déchiffrement

**b.1) Tampon d'entrée**

Le bus D transfère 8 bits à la fois, cependant le tampon d'entrée d'une largeur de 64 bits doit être divisé en huit parties de 8 bits chacune. Donc le tampon d'entrée sera formé de huit registres dont leur chargement se fait d'une manière séquentielle grâce à un registre à décalage (voir figure III.3.8). La sortie du registre à décalage est commandée par l'intermédiaire du signal LD_BO. Ce dernier sert à faire décaler sur la sortie du registre à décalage le un logique introduit par le signal IE depuis l'entrée LS_IN.

Une fois que l'ensemble des huit registres du tampon sont pleins, le signal ayant sélectionner le chargement du huitième registre traversera les deux bascules (FDCE) pour générer en sortie l'impulsion START qui déclenche l'opération de chiffrement ou de déchiffrement du bloc fonctionnel.

D'autres signaux nécessaires à la commande des différentes parties du tampon d'entrée seront décrits dans le paragraphe consacré aux modifications apportées au séquenceur du circuit DES pour l'adapter à cette nouvelle architecture.

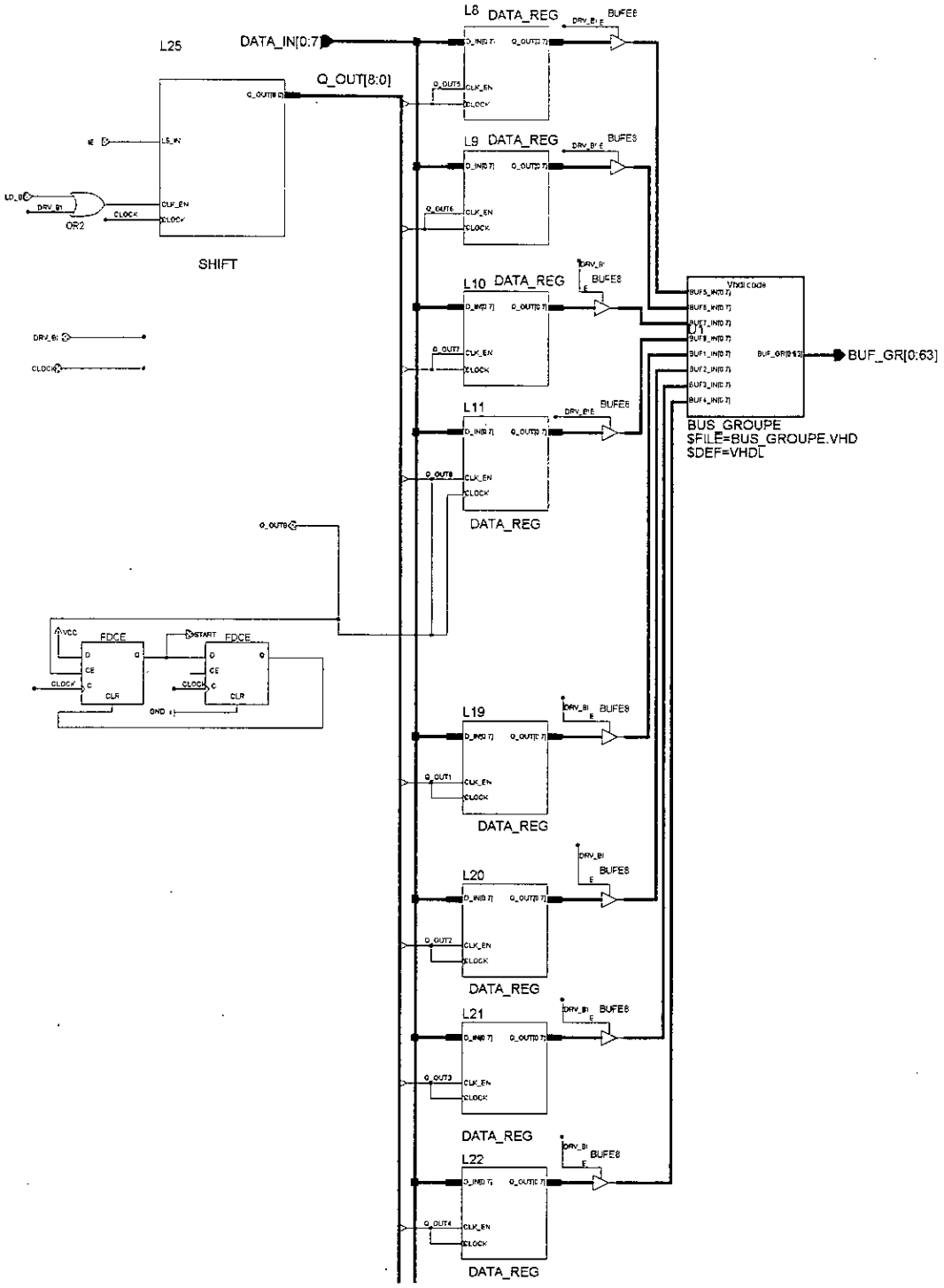



Figure III.3.8 : Tampon d'entrée

### b.2) Tampon de sortie

Les données traitées par le circuit DES quittent ce dernier sous forme de bloc de 64 bits. Ces données doivent être vidées sur le bus D d'une largeur de 8 bits. Une fois de plus un tampon de 64 bits de largeur composé de 8 registres montés en cascade doit être placé à la sortie du circuit DES.

A la fin d'une opération de chiffrement ou de déchiffrement, la donnée en sortie du circuit DES est chargée dans le tampon de sortie grâce au signal de commande LD_BO. Ce signal accède en même temps à l'entrée LS_IN du registre à décalage (Voir figure III.3.9) et représentera ainsi le bit le moins significatif de la sortie du registre à décalage.

Le vidage séquentiel des huit registres sera commandé par le signal DRV_BO provenant du séquenceur du cryptoprocesseur. Ce signal agira sur la sortie du registre à décalage qui commandera à son tour l'ouverture du tampon à trois états (BUF8) situé en sortie de chaque registre. Le signal d'état BOF indique si le tampon de sortie contient toujours des données à transférer sur le bus D. une fois que le huitième et dernier registre est vidé, le signal BOF sera remis à zéro.

### b.3) Séquenceur du circuit DES

Certaines modifications ont été apportées au séquenceur de la conception à base de l'architecture standard de l'algorithme DES. Les signaux ajoutés au séquenceur sont montés par la figure III.3.10. Ils concernent essentiellement la commande des tampons d'entrée/sortie. Ces signaux de commande sont conditionnés par les signaux d'états DE et START fournis par les tampons d'E/S, leurs rôles respectifs sont donnés par le tableau III.3.3.

La figure III.3.10 montre le diagramme temporel des signaux de commande. En plus des signaux de commande de la conception à base de la structure de base de l'algorithme DES, nous remarquons l'existence des signaux DRV_BI, RST_BI et OV dont leurs rôles respectifs sont donnés par le tableau III.3.3.

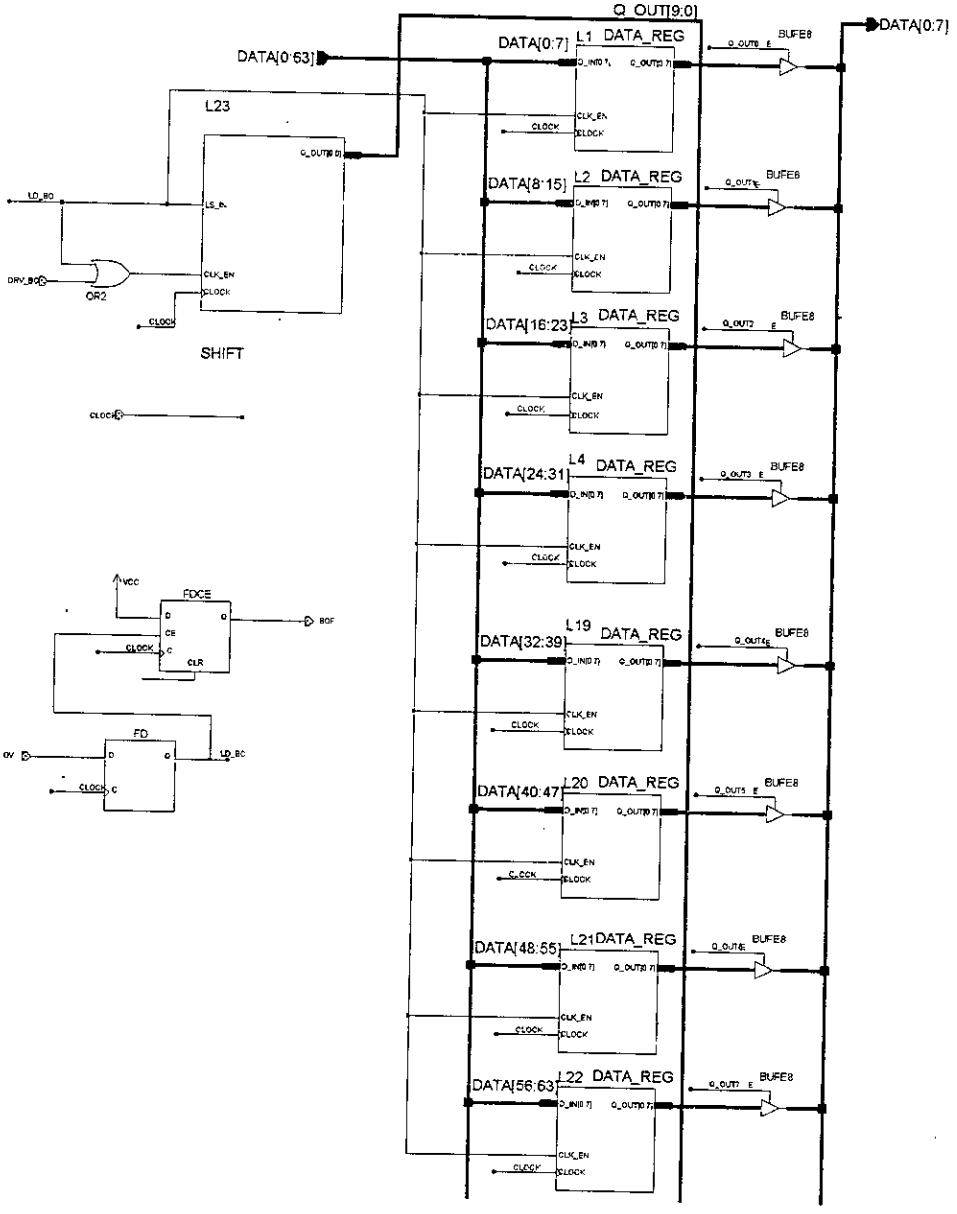



Figure III.3.9 : Tampon de sortie

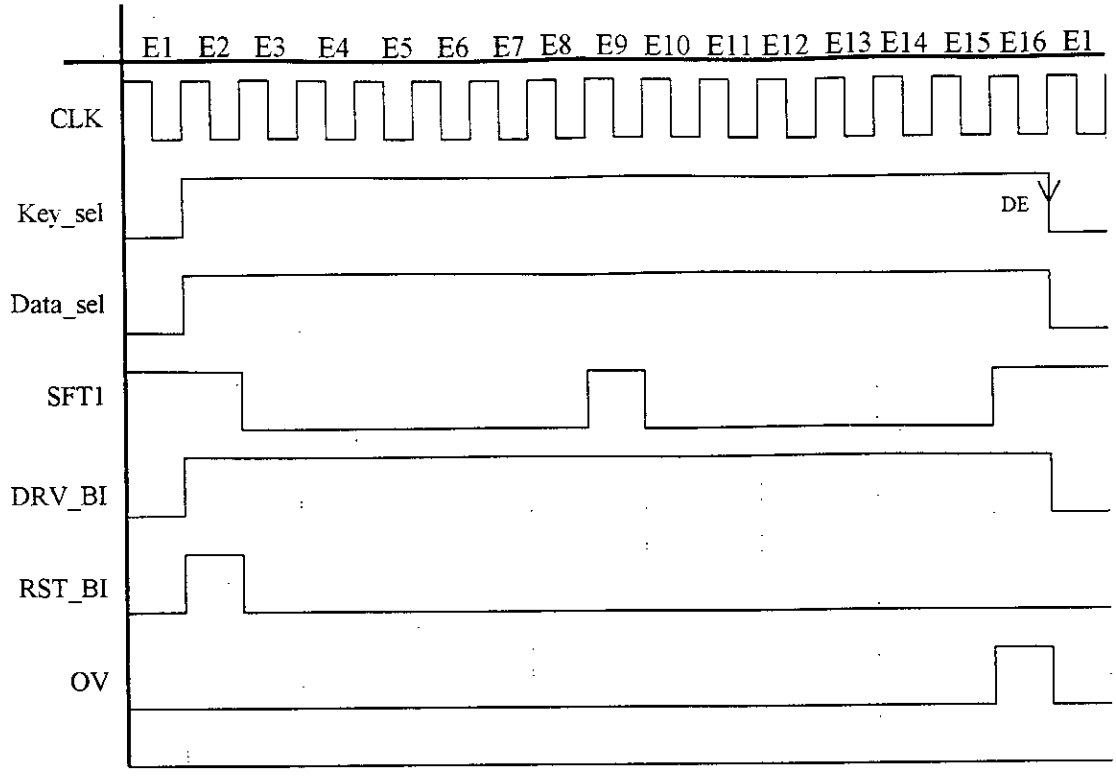



Figure III.3.10 : Signaux de commande du séquenceur du circuit DES

### III.3.3.2 Les registres et les bascules

#### a) L'ensemble des registres

Cet ensemble de registres à double port (voir figure III.3.11) est conçu à base d'une mémoire à double ports synchrone (composant RAM16X8D de la librairie disponible en standard dans le logiciel Xilinx) possède en entrée deux bus d'adresses (A_ADDR et B_ADDR) de 3 bits chacun. Ces deux bus d'adresses sont connectés aux 3 bits de poids faibles du registre d'instructions ([IR2..IR0]). Les entrées A_ADDR et B_ADDR sélectionnent le registre qui doit présenter son contenu sur les sorties respectives A_OUT et B_OUT. L'entrée A_ADDR sélectionne aussi le registre sur lequel va être écrite la valeur présente sur l'entrée A_IN provenant du bus D.

Pendant l'exécution des instructions, le cryptoprocesseur a souvent besoin d'un registre "brouillon" dans lequel il pourra stocker des valeurs et des adresses intermédiaires. Le registre R0 est destiné à cette tâche. Ce registre est adressé par l'intermédiaire des entrées R0A et R0B.

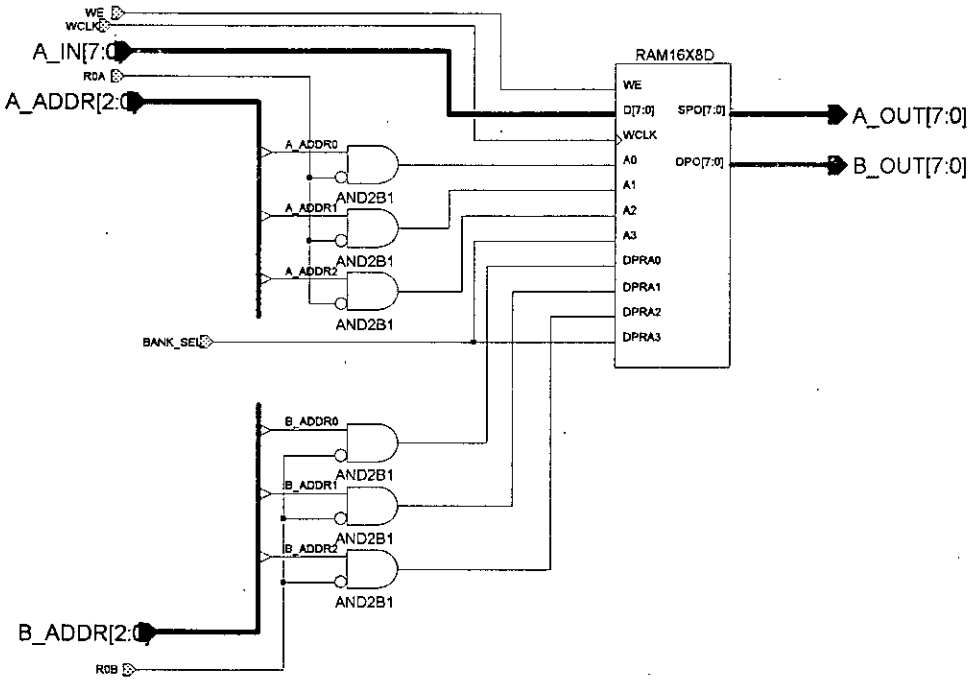



Figure III.3.11 : Circuit de l'ensemble de registres

Cet ensemble de seize registres est séparé en deux bancs de huit registres chacun. Chaque banc est réservé à un mode de fonctionnement du cryptoprocesseur (mode normal ou interruptible), et ainsi nous empêchons que les valeurs des registres du fonctionnement en mode normal soient altérées par les valeurs du fonctionnement en mode interruptible. La commutation entre ces deux bancs se fait grâce au signal de commande BANK_SEL. Ce signal agit sur le bit le plus significatif des deux adresses A_ADDR et B_ADDR.

La table de vérité de l'ensemble de registre est donné dans le tableau III.3.4.

**b) Circuit de sélection de l'adresse mémoire**

Le multiplexeur d'adresses est illustré par la figue 10.12. Il sélectionne une des trois adresses (registre interne, pointeur de pile (SP) ou compteur programme (PC)) présente sur l'une de ses trois entrées. Cette adresse forme les 8 bits de poids faible de l'adresse de la mémoire centrale et sert à sélectionner un emplacement mémoire dans l'un des segments PROGRAMME, DONNEES ou PILE.

Les entrées de commande SEL0 et SEL1 engendrent les 7 bits de poids forts de l'adresse mémoire. Cette portion de l'adresse sélectionne le segment à adresser (PROGRAMME, DONNEES ou PILE). Le tableau III.3.5 donne les valeurs de sorties du circuit sélecteur d'adresse en fonction des signaux de commande SEL0 et SEL1.

| SEL1 | SEL1 | [ADDR14..ADDR8] | [ADDR7..ADDR0] | Segment |
|---|---|---|---|---|
| 0 | 0 | 1111111 | [REGB7..REGB0] | DONNEES |
| 0 | 1 | 1111110 | [SP7..SP0] | PILE |
| 1 | 0 | 0000001 | [PC7..PC0] | Non utilisée |
| 1 | 1 | 0000000 | [PC7..PC0] | PROGRAMME |

Tableau III.3.5 : Sortie du circuit sélecteur d'adresse en fonction des signaux de commandes SEL0 et SEL1

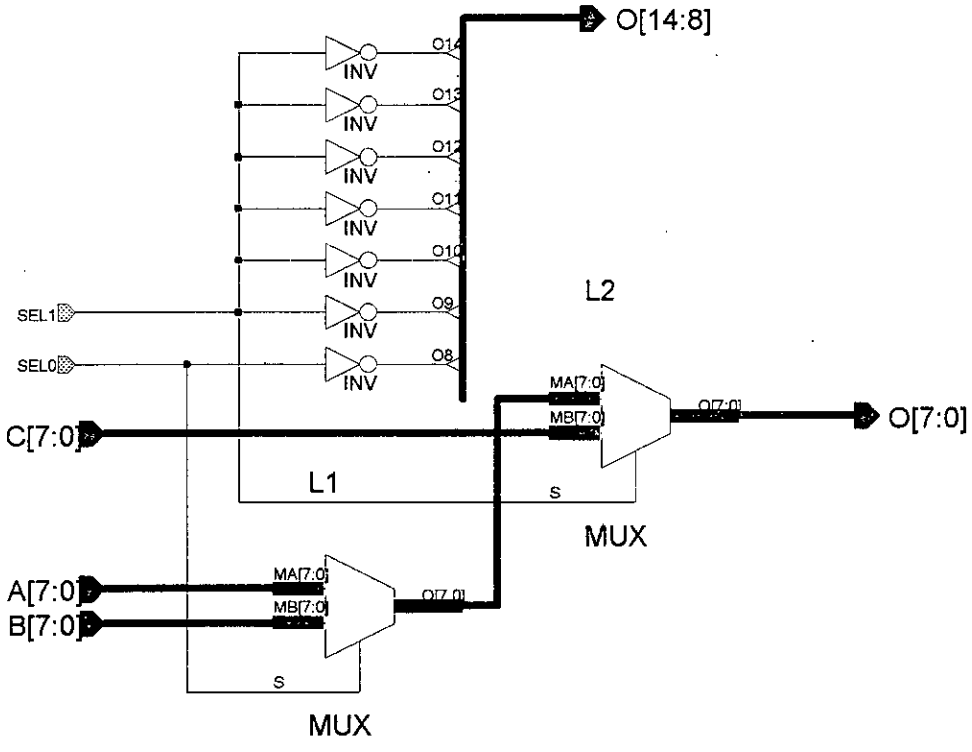



Figure III.3.12 : Circuit du sélecteur d'adresses

| INPUT | | | | | | | | OUTPUT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **A_ADDR[2:0]** | **B_ADDR[2:0]** | **LD_REG** | **R0A** | **R0B** | **A_IN[7:0]** | **BANK_SEL** | **CLK** | **[Rd] d=0,..,7** | **[Rd] d=8,..,15** | **A_OUT** | **B_OUT** |
| Rd | | | | | | 0 | ↑ | | | [Rd], d=0,..,7 | |
| | Rd | | | | | 0 | ↑ | | | | [Rd], d=0,..,7 |
| Rd | | 1 | | | A_IN[7:0] | 0 | ↑ | A_IN[7:0] | | | |
| X | | | 1 | | | 0 | ↑ | | | R0 | |
| | X | | | 1 | | 0 | ↑ | | | | R0 |
| Rd | | | | | | 1 | ↑ | | | [Rd], d=8,..,15 | |
| | Rd | | | | | 1 | ↑ | | | | [Rd], d=8,..,15 |
| Rd | | 1 | | | A_IN[7:0] | 1 | ↑ | | A_IN[7:0] | | |
| X | | | 1 | | | 1 | ↑ | | | R8 | |
| | X | | | 1 | | 1 | ↑ | | | | R8 |

Tableau III.3.4 : Table de vérité de l'ensemble des registre

### c) Le compteur programme

Le compteur programme est conçu à base d'un compteur rechargeable de huit bits de largeur. Le composant (CB8CLE) montré dans la figure III.3.13 est prit à partir de la liste standard des composants du logiciel Xilinx.

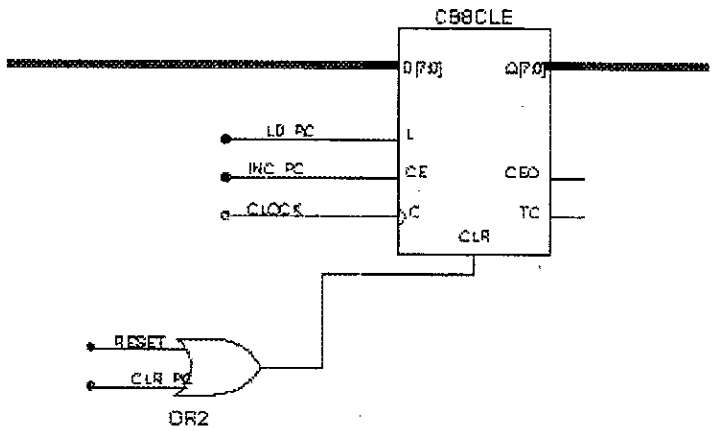



Figure III.3.13 : circuit du compteur programme

Le tableau III.3.6 donne la table de vérité du compteur programme.

| INPUT | | | | | | OUTPUT |
|---|---|---|---|---|---|---|
| **D[7:0]** | **LD_PC** | **INC_PC** | **RESET** | **CLR_PC** | **CLK** | **Q[7:0]** |
| D[7:0] | 1 | 0 | 0 | 0 | ↑ | D[7:0] |
| X | 0 | 1 | 0 | 0 | ↑ | PC ← PC + 1 |
| X | X | X | 1 | X | X | 00000000 |
| X | X | X | X | 1 | X | 00000000 |

Tableau III.3.6 : Fonctionnement du circuit compteur programme

### d) Pointeur de pile

La figure III.3.14 montre que le pointeur de pile SP (identifié par la référence SP0) est à base d'un compteur incrémentation/décrémentation, rechargeable d'une largeur de huit bits (CC8CLED).

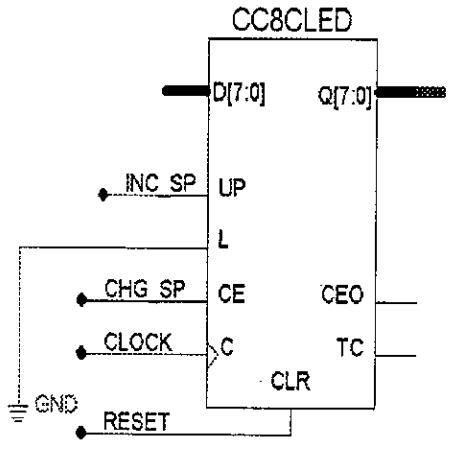



Figure III.3.14 : circuit du pointeur de pile

Son principe de fonctionnement est donné par le tableau III.3.7.

| INPUT | | | | OUTPUT |
|---|---|---|---|---|
| **CHG_SP** | **INC_SP** | **RESET** | **CLK** | **Q[7:0]** |
| 1 | 1 | 0 | ↑ | SP ← SP + 1 |
| 1 | 0 | 0 | ↑ | SP ← SP - 1 |
| X | X | 1 | X | 00000000 |

Tableau III.3.7 : Fonctionnement du circuit pointeur de pile

### e) Registre d'instructions

Le registre d'instruction (module IR0) est directement connecté au bus DATA. Par conséquent, le registre d'instruction peut être chargé avec n'importe quelle donnée ce trouvant sur les broches [DATA7..DATA0] à condition que le signal LD_IR=1. La figure III.3.15 montre le circuit du pointeur de pile.

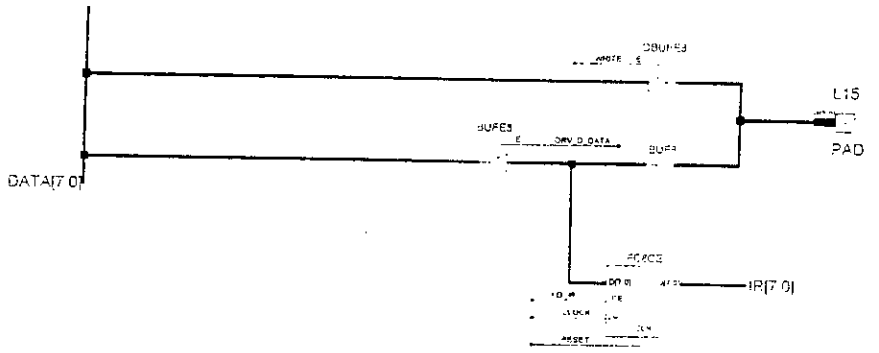



Figure III.3.15 : Circuit du registre d'instructions

### f) Bloc de traitement d'interruption

La figure III.3.16 montre le circuit de détection et d'enregistrement de l'état de l'interruption. Un état haut sur l'entrée d'interruption (INT) est chargé dans la première bascule D. Ensuite, il se propage à travers la bascule INTRPT_DFF0 et apparaît sur la sortie INTRPT au prochain front montant de l'horloge. Ceci à pour but de synchroniser l'interruption avec l'horloge du cryptoprocesseur.

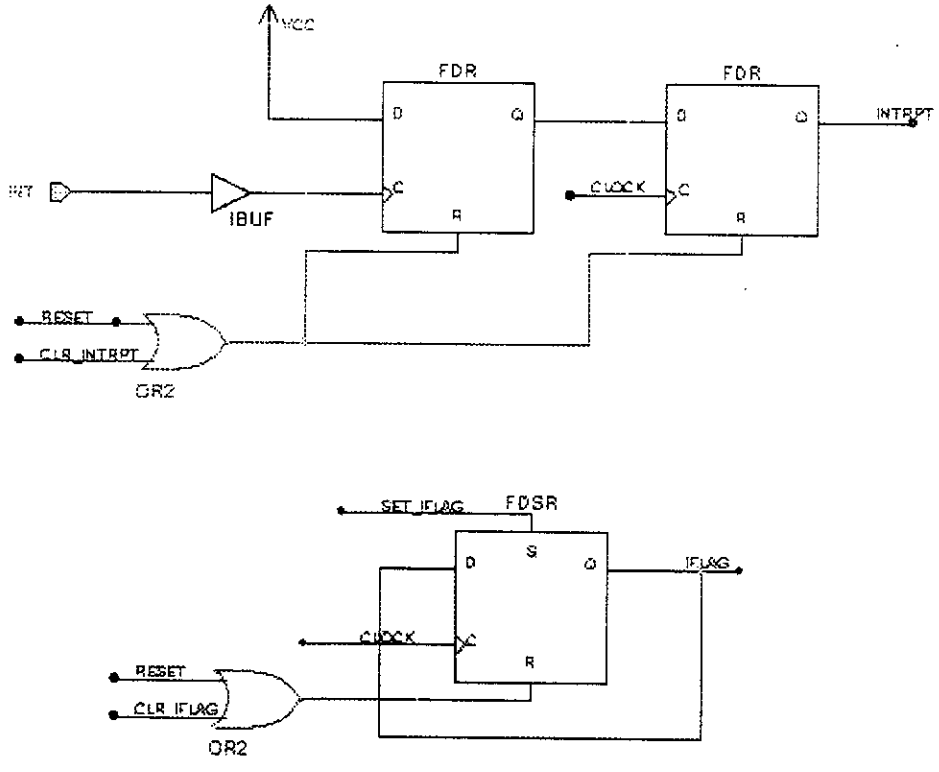



Figure III.3.16 : Circuit du bloc de traitement d'interruptions

En plus, de la bascule d'enregistrement de la présence d'une interruption, le cryptoprocesseur a aussi besoin d'une autre bascule, bascule IFLAG0, qui lui indique qu'un programme prioritaire est déjà en cours d'exécution. La bascule IFLAG0 est fixée à l'état logique 1 quand le signal SET_IFLAG=1. Le signal de sortie IFLAG de la bascule est utilisé pour commuter sur les huit registres dédie aux programmes de traitements des interruptions. Aussi, les signaux IFLAG et INTRPT sont des entrées du séquenceur. Si INTRPT=1 et IFLAG=0, le séquenceur doit changer le flow d'instructions par le lancement du programme de traitement des interruptions. Mais, si INTERPT=1 et IFLAG=1, le cryptoprocesseur ignore l'interruption en attente car il est traite déjà une interruption.

Le contenu des bascules des interruptions est effacé lors de la mise à 1 des signaux CLR_INTRPT, CLR_IFLAG et RESET.

### g) Port d'entrée/sortie

Le circuit du port d'entrée/sortie est donnée par la figure III.3.17. Ce port est construit à base de seize bascules dont huit d'entre elles sont dédiées au port d'entrée et les huit autres sont réservées au port de sortie. Ces seize bascules sont situées dans les blocs d'E/S (IOBs) [11], qui se trouvent le long de la périphérie du composant FPGA (modules, OFDEX8 et IFD8). Donc, ces bascules ne consomment pas de ressources du champ de CLBs du composant FPGA.

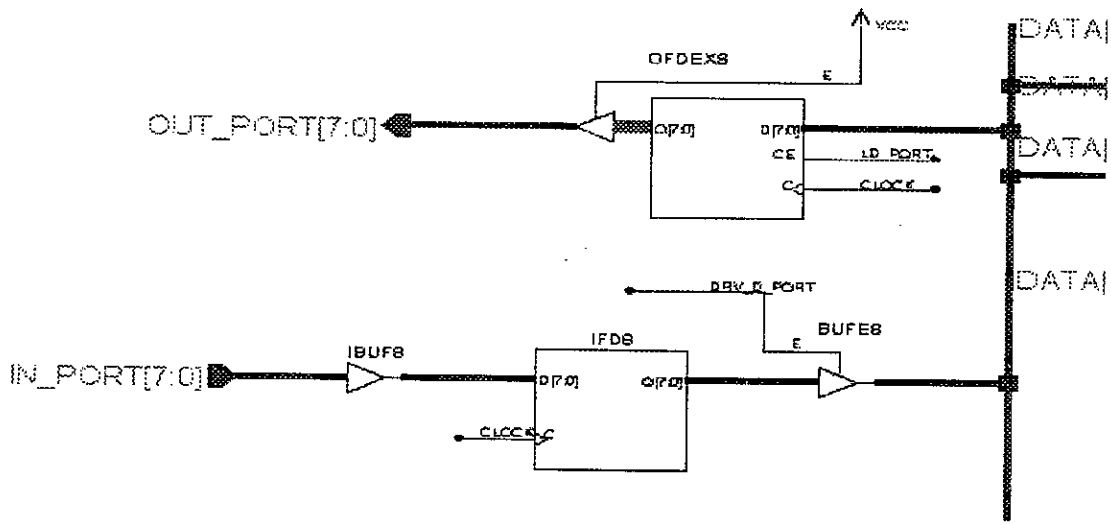



Figure III.3.17 : Circuit du port d'E/S

Le principe de fonctionnement du port d'E/S est donné par le tableau ci-dessous.

| INPUT | | | | | OUTPUT | |
|---|---|---|---|---|---|---|
| **DATA[7 :0]** | **IN_PORT[7 :0]** | **DRV_D_PORT** | **LD_PORT** | **CLK** | **OUT_PORT[7 :0]** | **DATA[7 :0]** |
| DATA[7 :0] | | | 1 | ↑ | DATA[7 :0] | |
| | IN_PORT[7 :0] | 1 | | ↑ | | IN_PORT[7 :0] |

Tableau III.3.8 : Fonctionnement du port d'E/S

### III.3.3.3 Les bus d'interconnexions

Le bus de données bidirectionnel, reliant le cryptoprocesseur avec la mémoire est montré par la figure III.3.18. Les données sur le bus D quittent les broches de sortie [DATA7..DATA0] lorsque le signal WRITE=1. Ceci valide les sorties du tampon à trois états (OBUF8). Les données peuvent accéder au cryptoprocesseur en mettant le signal DRV_D_DATA à un. Ceci valide l'entrée des tampons à trois états (BUFE8) ; et ainsi les données sur le bus DATA apparaissent sur le bus D interne.

Le circuit globale du cryptoprocesseur est donné par la figure III.3.18, où nous remarquons en plus des divers circuits sus cités; l'existence des circuits pour piloter les entrées de commande de la mémoire centrale. Ces circuits sont montrés dans la portion haute à droite de la figure III.3.18. La mémoire est constamment valide, par le fait de forcer à la masse le signal de sélection de composant (CS_). Le signal sortie valide de la mémoire (OE_) est mis à l'état bas lors d'une opération de lecture. Le signal (WE_) valide les opérations d'écriture de la mémoire.

## III.3.4 Conception fonctionnelle de la partie commande

Après étude et organisation des signaux de contrôle gérant le fonctionnement du cryptoprocesseur, nous avons proposé une partie de contrôle qui automatise le déroulement des différentes opérations, en assurant la synchronisation des étapes de calcul ainsi que la coordination entre les modules constituant l'architecture globale.

Le séquenceur reçoit en entrée quatorze signaux d'états, à savoir : les indicateurs de retenue et de zéro, les deux signaux d'états du bloc de traitement d'interruption, les 8 bits du registre d'instructions, les signaux d'états des tampons du bloc fonctionnel de chiffrement/déchiffrement, l'horloge CLK et le signal RESET. Il fournie en sortie 33 signaux pour le contrôle des différentes circuits constituants le cryptoprocesseur.

Une vue synthétique du comportement du séquenceur est donnée par l'organigramme de la figure III.3.19 Cet organigramme résume les différentes étapes de fonctionnement du cryptoprocesseur.

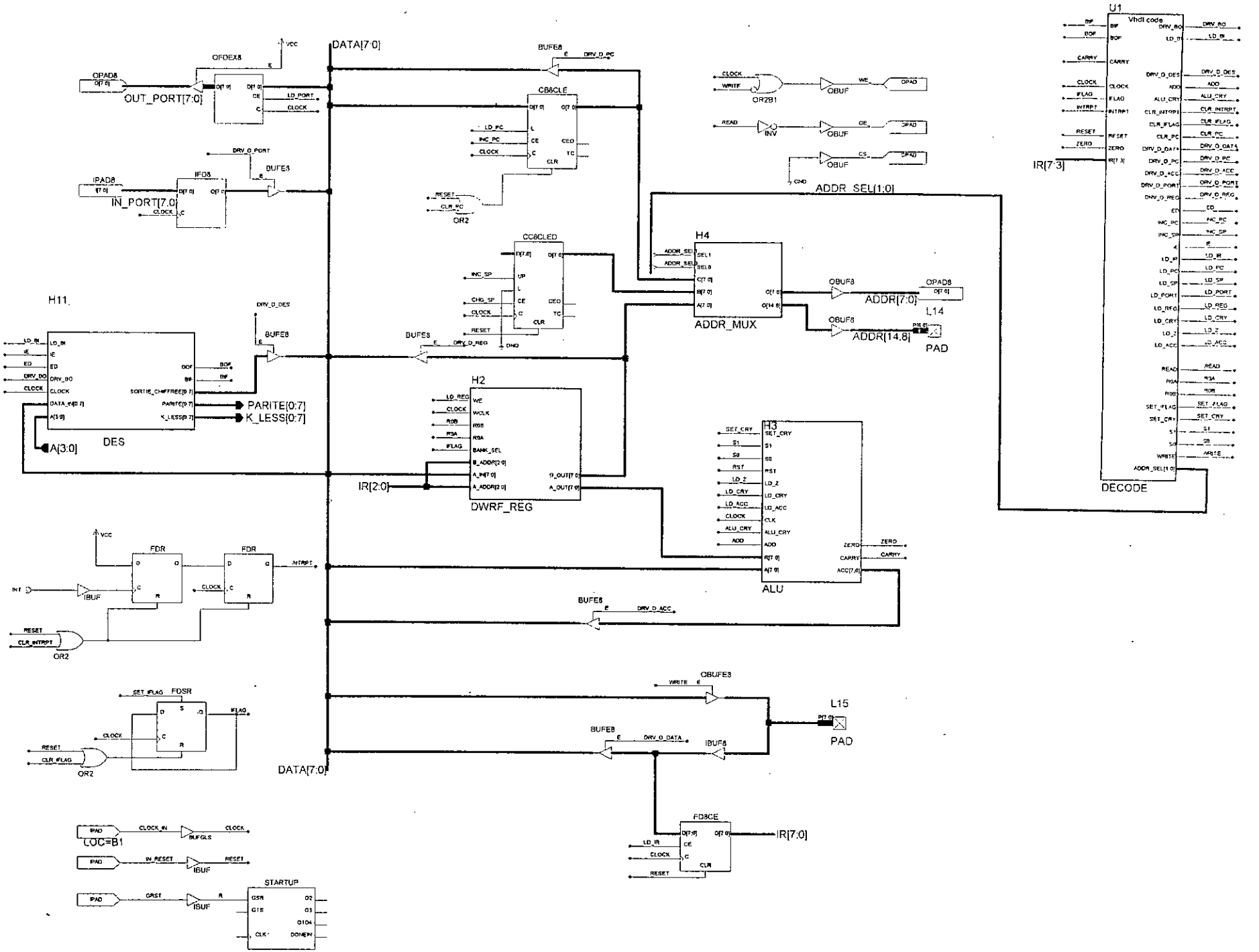
Le module décodeur d'instructions (voir figure III.3.18) a été décrit en code VHDL ensuite inséré dans la description schématique globale du cryptoprocesseur sous forme d'un fichier netlist synthétisé.

## III.3.5 Conclusion

Nous avons présenté les modèles structurels des deux parties chemin de données et séquenceur avec les facilités de conception qui sont rendues possibles par l'adoption de l'approche de conception Down-Top.

Nous avons terminé notre étude par l'implémentation du modèle structurel du cryptoprocesseur sur le circuit FPGA XC4013XL-3BG256. Le modèle physique ainsi obtenu est présenté dans l'annexe A.6.

Afin de valider les performances données par l'analyseur temporel statique et pour confirmer les résultats obtenus par les simulations fonctionnelles. Nous avons procédé à des simulations temporelles du cryptoprocesseur, selon plusieurs mode de fonctionnement, à savoir : fonctionnement en mode normal et en mode interruptible, fonctionnement selon le mode d'adressage et en dernier une simulation d'une opération de chiffrement d'un bloc de données, stocké dans le segment DATA de la mémoire. Toutes ces simulations sont données respectivement dans les annexes C, D et E.


DATA[7:0]
OUT_PORT[7:0]
IN_PORT[7:0]
OPAD8
IPAD8
OFDEX8
IFD8
BUFE8
CB8CLE
OR2B1
INV
OBUF
OR2
ADDR_SEL[1:0]
H4
ADDR_MUX
CC8CLED
ADDR[7:0]
ADDR[14:8]
L14
PAD
H11
DES
A[3:0]
PARITE[0:7]
K_LESS[0:7]
H2
DWRF_REG
IR[2:0]
H3
ALU
U1
Vhdl code
DECODE
IR[7:3]
FDR
IBUF
INTRPT
FDSR
IFLAG
OBUFE8
L15
BUFE8
FD8CE
IR[7:0]
LOC=B1
IBUFG
STARTUP

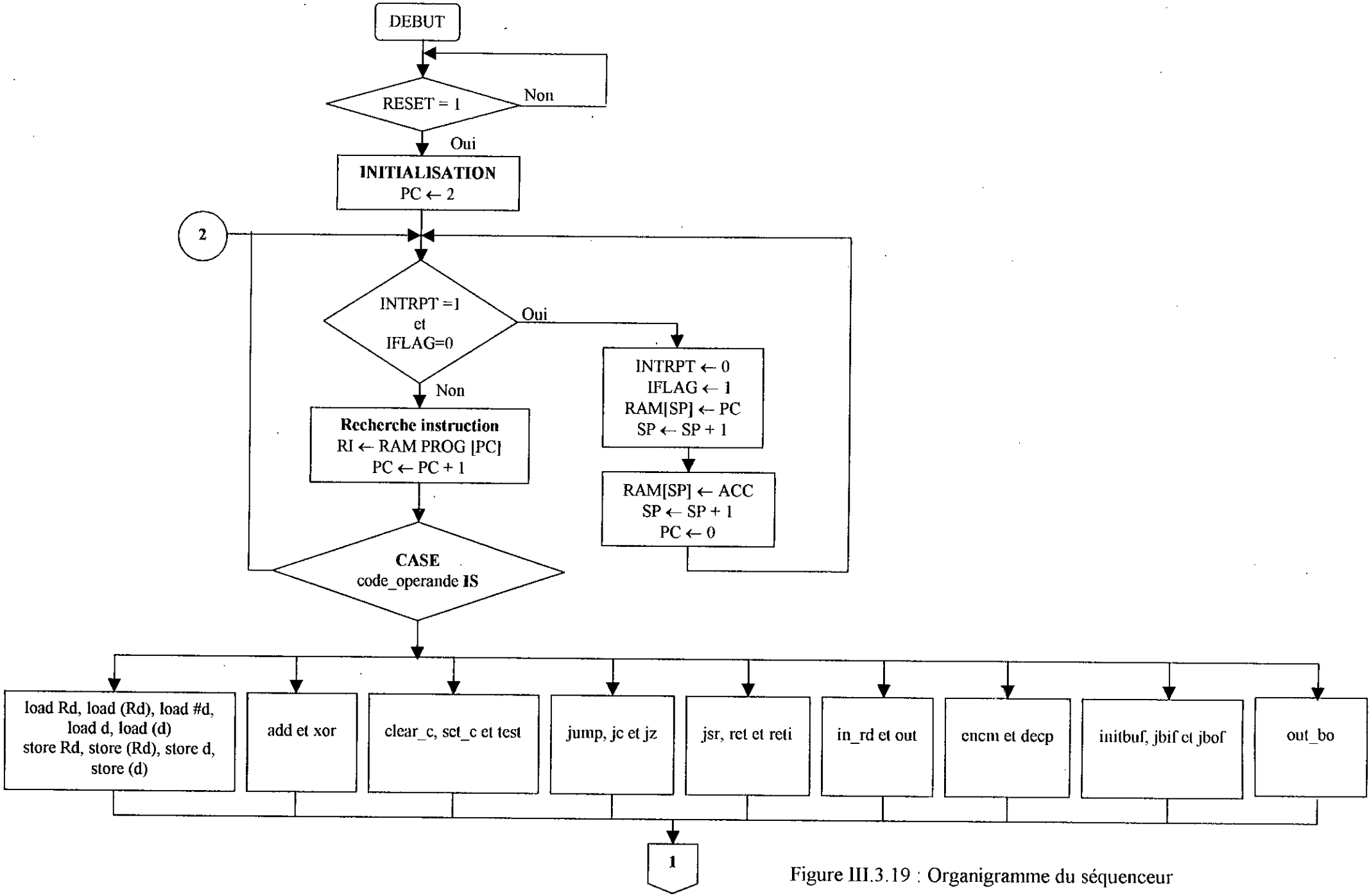



Figure III.3.19 : Organigramme du séquenceur

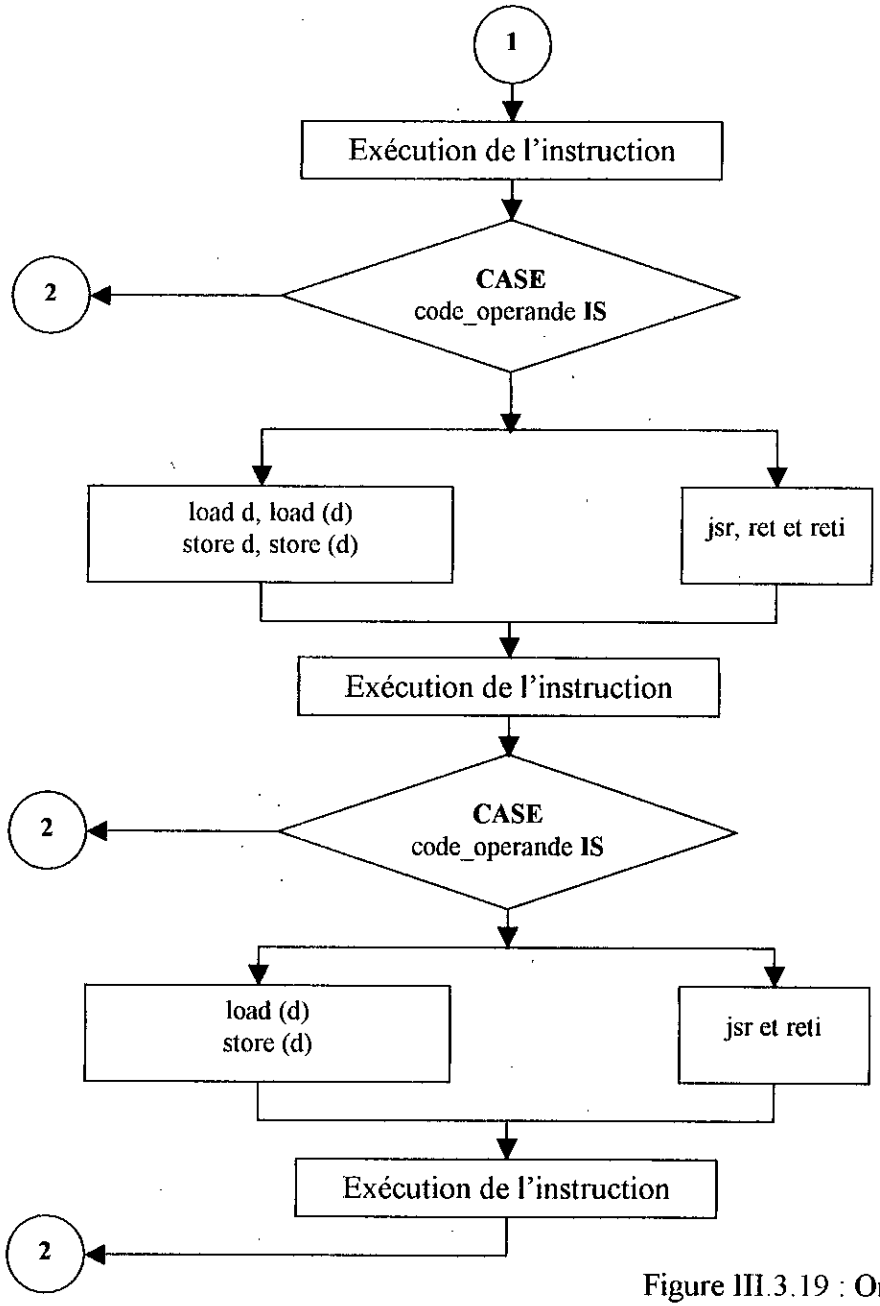



Figure III.3.19 : Organigramme du séquenceur (suite)

# Chapitre IV

# Mise en œuvre et évaluation

## IV.1 Introduction

Nous avons implémenté multiples versions de chaque option architecturale listée dans le tableau II.2.1, afin d'évaluer leur performance. Toutes les conceptions ont été implémentées sur le même composant FPGA XC4013XL-3BG256.
Dans les paragraphes qui vont suivre, nous comparerons les différentes conceptions entre elles. Dans la plupart des cas, nos conceptions seront comparées à la conception de référence DES_VER1 qui nous servira comme modèle de référence.

Après l'implémentation des différentes options architecturales de l'algorithme DES, nous présenterons les résultats obtenus par l'implémentation du microprocesseur (cryptoprocesseur sans le module de chiffrement/déchiffrement) et ensuite ceux obtenus par l'implémentation du cryptoprocesseur.

L'unité CLB (combinatorial logic block) [11] utilisée par le fabriquant Xilinx pour mesurer la quantité de ressources logiques dans le composant FPGAs, sera utilisée pour évaluer la quantité de ressources logiques consommée par une conception donnée. L'abréviation ULC signifie Unité Logique Combinatoire (voir le sous-chapitreI.4)

## IV.2 Algorithme DES avec une architecture en boucle expansée

Nous avons implémenté deux versions à base de la structure en boucle expansée : DES_VER2.1 et DES_VER2.2. La conception DES_VER2.1 contient deux unités logiques combinatoires (ULCs, voir paragraphe II.2.3) et par conséquent une opération de chiffrement d'un bloc de donnée tient sur 8 cycles d'horloge alors que, la conception DES_VER2.2 qui contient quatre ULCs fournira le résultat d'une opération de chiffrement après 4 cycles d'horloge. Ces deux conceptions seront toutes les deux comparées à la conception DES_VER1. Le tableau IV.1 englobe tous les résultats obtenus par les différentes conceptions.

| Conception | CLB (utilisés) | Nombre d'ULC | CLB par ULC | CLK Min (ns) | Débit par ULC (Mbit/s) | Débit (Mbit/s) | Performance relative (%) | Consommation supplémentaire de ressource (%) |
|---|---|---|---|---|---|---|---|---|
| DES_VER1 | 195 | 1 | 195 | 36.4 | 109.8 | 109.8 | - | - |
| DES_VER2.1 | 312 | 2 | 156 | 53.2 | 75.1 | 150.3 | 37 | 60 |
| DES_VER2.2 | 551 | 4 | 138 | 85.5 | 46.8 | 187.1 | 25 | 76 |

Tableau IV.1 : Comparaison des architectures en boucle expansée

La conception DES_VER2.1 offre un débit 37% plus élevé par rapport à la version de base DES_VER1 alors que les ressources consommées (en CLBs) ont augmentés de 60%. La conception DES_VER2.2 est seulement de 25% plus rapide que la conception DES_VER2.1, donc le débit a seulement augmenté d'un peu plus que la moitié par rapport à la première structure en boucle expansée. Les ressources consommées ont augmentés de 76%.

Le nombre de CLBs divisé par le nombre d'ULCs montre que la quantité de ressources logiques consommées, par le simple fait de procéder à la concaténation d'ULCs, augmente d'une façon presque constante. Le débit divisé par le nombre d'ULCs montre que le débit d'une ULC, dans la conception DES_VER2.2 est égale à la moitié du débit de la conception DES_VER1.

D'après l'évaluation des résultats obtenus, nous pouvons conclure que la quantité de ressources logiques consommées augmentent linéairement contrairement au débit.

## IV.3 Algorithme DES avec une architecture en pipeline

Nous avons implémentés deux conceptions à base d'une structure en pipeline, DES_VER3.1 et DES_VER3.2. La conception DES_VER3.1 contient deux ULCs et par conséquent elle possède un pipeline à deux étages. La conception DES_VER3.2 contient quatre ULCs et par conséquent elle possède un pipeline à quatre étages. Dans les deux cas, une opération de chiffrement d'un bloc de données tient sur seize cycles d'horloge.

En réalité, seule la conception DES_VER3.1 qui a pu être implémentée. La seconde conception DES_VER3.2 n'a pas pu être implémentée car les ressources consommées par cette conception dépassent les ressources disponibles sur le composant FPGA XC4013XL-3BG256. Ce composant dispose d'un nombre de CLBs égal à **576**.

Le tableau IV.2 résume les résultats obtenus.

| Conception | CLB (utilisés) | Nombre d'ULCs | CLB par ULC | CLK Min (ns) | Débit par ULC (Mbit/s) | Débit (Mbit/s) | Performance relative (%) | Consommation supplémentaire de ressource (%) |
|---|---|---|---|---|---|---|---|---|
| DES_VER1 | 195 | 1 | 195 | 36.4 | 109.8 | 109.8 | - | - |
| DES_VER3.1 | 327 | 2 | 164 | 41.6 | 90.5 | 180.9 | 65 | 68 |
| DES_VER3.2 | 589 | 4 | 147 | - | - | - | - | 80 |

Tableau IV.2 : Comparaison des architectures en pipeline

La conception DES_VER3.1 offre un débit 65% plus élevé par rapport à la version de base DES_VER1 alors que les ressources consommées (en CLBs) ont augmentés de 68%. La conception DES_VER3.2 possède une consommation supplémentaire de 80% par rapport à la version DES_VER3.1.

Le débit divisé par le nombre d'ULCs montre que ce rapport reste presque constant pour toutes les conceptions. La diminution du débit dans la conception DES_VER3.1 est due au nombre limité de ressources d'interconnexions disponibles dans le composant FPGA XC4013XL-3BG256. Ce phénomène engendre une conception moins performante.

La quantité de ressources logiques consommés diminue si nous augmentons le nombre de pipeline. Ceci est du d'une part à l'unité de contrôle qui ne va pas être plus compliquée et d'autre part aux multiplexeurs qui sont implémentés qu'une seule fois.

Il est intéressant de comparer la conception en pipeline avec les conceptions à structure en boucle expansée. Nous remarquons que le débit de la conception DES_VER3.1 est de 20% plus grand que celui de la conception DES_VER2.1 alors que les ressources consommées en CLBs ont augmentés uniquement de 4%. Dans la version DES_VER3.2, la consommation en ressources est de 6% plus grande par rapport à la conception DES2.2.

L'analyse des performances de l'implémentation de la structure en pipeline s'est restreinte uniquement à la conception possédant une structure à deux (02) étages. Ceci est insuffisant pour évaluer les performances de cette option architecturale. Mais d'autres études [36] ont abouti aux résultats suivants : la quantité de ressources logiques consommées et le débit augmentent linéairement avec l'accroissement du nombre d'étages du pipeline.

## IV.4 Algorithme DES avec une architecture combinée (pipeline et en boucle expansée)

La conception qui regroupe les deux options architecturales à savoir, l'architecture à structure en boucle expansée et l'architecture à structure en pipeline, a été implémentée sous la version DES_VER4. Cette conception contient 4 ULCs ; deux ULCs dans chacun des deux pipelines. Le tableau IV.3 compare cette conception avec les versions DES_VER3.1 et DES_VER2.1.

| Conception | CLB (utilisés) | Nombre d'étages du pipeline | CLK Min (ns) | Débit par pipeline (Mbit/s) | Débit (Mbit/s) |
|---|---|---|---|---|---|
| DES_VER4 | 526 | 2 | 53.3 | 133.4 | 266.8 |
| DES_VER3.1 | 327 | 2 | 41.6 | 90.5 | 180.9 |
| DES_VER2.1 | 312 | 1 | 53.2 | 150.3 | 150.3 |

Tableau IV.3 : Comparaison de l'architecture combinée avec deux autres architectures en boucle expansée et en pipeline

Il n'est pas facile de comparer la conception combinée avec les deux autres conceptions DES_VER3.1 et DES_VER2.1. La période d'horloge minimum montre que dans la conception DES_VER3.1, le temps nécessaire aux deux ULCs pour exécuter un bloc de données est de 28% plus petit par rapport à la conception DES_VER4. Les deux conceptions DES_VER2.1 et DES_VER4 possèdent le même temps d'exécution alors qu'en réalité la conception DES_VER4 devrait en avoir un temps d'exécution plus petit. Cette anomalie est due essentiellement au nombre limité de ressources d'interconnexions disponibles dans le composant FPGA XC4013XL-3BG256. Engendrant ainsi la dégradation des performances de la conception DES_VER4.

## IV.5 Résumé des résultats

Le tableau IV.4 résume les résultats de toutes les conceptions implémentées. La plus rapide des implémentation à base d'une structure en boucle expansée, est la conception **DES_VER2.2** avec un débit de **187.1 Mbit/s**. Tandis que la plus performante des implémentations possédants une structure en pipeline, est la conception **DES_VER3.1** avec un débit de **180.9 Mbit/s**.

| Conception | Nombre CLB/utilisés | Nombre d'ULCs | CLB par ULC | CLK Min (ns) | Débit par ULC (Mbit/s) | Débit (Mbit/s) |
|---|---|---|---|---|---|---|
| DES_VER1 | 195 | 1 | 195 | 36.4 | 109.8 | 109.8 |
| DES_VER2.1 | 312 | 2 | 156 | 53.2 | 75.1 | 150.3 |
| DES_VER2.2 | 551 | 4 | 138 | 85.5 | 46.8 | **187.1** |
| DES_VER3.1 | 327 | 2 | 164 | 41.6 | 90.5 | **180.9** |
| DES_VER3.2 | 589 | 4 | 147 | - | - | - |
| DES_VER4 | 526 | 4 | 132 | 53.3 | 66.7 | 266.8 |

Tableau IV.4 : Tableau récapitulatif de toutes les architectures implémentées

## IV.6 Les résultats de l'implémentation du microprocesseur

Avant d'implémenter le cryptoprocesseur nous avons jugé utile d'implémenter le microprocesseur seul (cryptoprocesseur sans le module de chiffrement/déchiffrement). La principale raison est d'une part le test du fonctionnement du microprocesseur, l'évaluation de ces performances et d'autre part voir les conséquences sur les performances en rajoutant le module chiffrement/déchiffrement.
Cette conception a été implémentée sur le circuit FPGA XC4013XL-3BG256. Les résultats obtenus sont résumés dans le tableau ci-dessous.

| Conception | Nombre CLB/utilisés | Consommation de ressource (%) | Fréquence (Mhz) |
|---|---|---|---|
| Microprocesseur | 109 | 18 | 21.5 |

Tableau IV.5 : Résultats de l'implémentation du microprocesseur

## IV.7 Les résultats de l'implémentation du Cryptoprocesseur

Le cryptoprocesseur a été aussi implémenté sur le circuit cible XC4013XL-3BG256. La consommation en ressources et la fréquence de travail du cryptoprocesseur sont données dans le tableau IV.6.

| Conception | Nombre CLB/utilisés | Consommation de ressource (%) | Fréquence (Mhz) |
|---|---|---|---|
| Cryptoprocesseur | 346 | 60 | 20.5 |

Tableau IV.6 : Résultats de l'implémentation du cryptoprocesseur

## IV.8 Conclusion

La conception DES_VER4 est considérée comme étant la plus rapide des implémentations avec un débit de **266.8 Mbit/s**. Mais la conception à quatre pipeline DES_VER3.2 devrait normalement donner un meilleur débit.

Dans le cas de l'implémentation de l'algorithme DES avec une structure en boucle expansée, nous pouvons conclure que la quantité de ressources logiques consommées augmentent linéairement contrairement au débit. Alors que dans le cas de l'implémentation de l'algorithme DES avec une structure en pipeline, la quantité de ressources logiques consommées et le débit augmentent linéairement avec l'accroissement du nombre d'étages du pipeline

Le fait d'avoir rajouté le module chiffrement/déchiffrement au microprocesseur n'a pas dégradé, d'une manière notable, ses performances.

# Conclusion

Durant notre travail et pendant les phases d'implémentation des différentes conceptions nous avons formulé certaines recommandations afin d'arriver à réaliser une implémentation, la plus performante que possible de l'algorithme DES.

- Pour un maximum de performance, les tables-S doivent être implémentées sur des modules ROM. La rapidité d'exécution des tables-S influe considérablement sur la performance de la conception globale.
- Les blocs fonctionnels de permutation et d'expansion sont implémentés en utilisant seulement les ressources d'interconnexions.
- Les registres à décalage peuvent être implémentés en utilisant uniquement des ressources d'interconnexions associées à des multiplexeurs (éléments disponibles dans la structure de base d'un CLB).
- L'utilisation des modules LogiBLOX facilitent d'une part la conception et d'autre part, ils sont déjà optimisés selon le composant FPGA cible.

La subdivision de la structure de base de l'algorithme DES sous la forme de blocs fonctionnels simplifie considérablement la modification de la conception. Par conséquent, pour la création de nouvelles options architecturales de l'algorithme DES nous avons juste besoin d'ajouter ou de soustraire certains blocs fonctionnels et ainsi concevoir une nouvelle logique de commande.

Le chemin de données du cryptoprocesseur a été conçu selon l'approche Down-Top (du Bas vers le Haut). L'adoption de cette approche facilite considérablement la réalisation du chemin de données global et permet de vérifier séparément le fonctionnement de chaque bloc fonctionnel.

Nous avons implémenté toutes les conceptions sur un composant FPGA de XILINX. Les résultats les plus importants obtenus peuvent être énumérés comme suit:

**a) Algorithme DES**

- **Débit maximum** : Nous avons atteint un débit de 266.8 Mbit/s.
- **Comparaison des performances** : Si nous comparons la vitesse de l'implémentation de l'algorithme DES sur un circuit ASIC (1600 Mbit/s) [1] et sur une solution logicielle (12 Mbit/s) [1] ; avec notre meilleur résultat de 266.8 Mbit/s nous pouvons conclure que le rapport de vitesse entre la solution logicielle et le circuit FPGA est de 22.3, et entre le circuit FPGA et le circuit ASIC il est de 6.

Les résultats obtenus par l'implémentation des différentes options architecturales de l'algorithme DES ont été évalués d'une façon très détaillée. Les points les plus importants à retenir sont les suivants :

- **Boucle expansée** : Avec la première expansion nous avons pu augmenter le taux de chiffrement de 37% et une utilisation supplémentaire de ressources logiques de 60%. Avec la seconde expansion nous avons seulement gagner un débit supplémentaire de 25% par rapport à la première expansion et en plus un pourcentage de 76% de ressources consommées.
  **Conclusion** : La quantité de ressources logiques consommées augmentent linéairement, par contre la vitesse augmente plus lentement.

- **Pipeline** : Avec un pipeline à deux (02) étages nous avons gagné un débit supplémentaire de 65% et en plus 68% de ressources consommées. Avec un pipeline à quatre (04) étages nous prévoyons un débit qui dépasse les 110% par rapport à celui obtenu avec le pipeline à deux étages et nous avons aussi une consommation en ressources logiques qui augmente de 80%.
  La quantité de ressources logiques consommées et la vitesse augmentent linéairement.

- **Conception combinée** : Le résultat de cette implémentation est l'obtention d'un meilleur débit par la combinaison des deux conceptions suscitées.

La conception ayant une structure en boucle expansée n'a pas donné le plus grand débit mais l'avantage majeur de cette conception est la possibilité de l'utiliser avec n'importe quel mode de fonctionnement de l'algorithme DES (voir paragrapheI.4.4). Les conceptions à base d'une structure en pipeline sont plus rapides mais peuvent être utilisées uniquement dans les modes qui ne sont pas basés sur le principe de la contre réaction. Les conceptions possédantes une structure en pipeline peuvent cependant être utilisées dans les cryptosystèmes qui fonctionnent en mode ECB. Ceci est aussi valable pour les conceptions combinées du fait qu'elles contiennent un pipeline.

**b) le processeur**

- **Fréquence maximum** : le cryptoprocesseur peut atteindre une fréquence de travail de 20.5 MHz sur une cible XC4013XL-3BG256

- **Comparaison des performances :** Si nous comparons les performances obtenues par l'implémentation du cryptoprocesseur, d'une part avec le résultat de la firme CAST, Inc [14] (101 MHz) et d'autre part avec celui des travaux de thèse de Põldre J [13] (25 MHz), nous pouvons déduire que le rapport de fréquence entre la conception professionnelle (firme CAST, Inc) et notre conception est de 5, alors qu'entre la conception du monde universitaire (thèse de master de Põldre J [13]) et notre conception ce rapport est égal à 1.2.

Nous avons seulement implémenté l'algorithme DES en mode ECB. Il serait donc très intéressant d'exploiter cette conception afin qu'elle supporte tous les modes de fonctionnement définis pour l'algorithme DES, les implémenter sur le même composant FPGA avec des objectifs de performance en vitesse plus intéressantes.

Les fonctions cryptographiques peuvent être étendues. En plus, de la fonction de chiffrement de données assurée par l'algorithme DES, nous envisageons de faire intégrer les fonctions cryptographiques suivantes :

- La fonction chiffrement de données ne se fera plus uniquement par l'algorithme DES mais par le choix entre trois algorithmes cryptographiques à savoir :
  - DES (Data Encryption Standard),
  - IDEA (International Data Encryption Algorithm),
  - AES (Advanced Encryption Standard), successeur de l'algorithme DES.
- L'échange de clefs sera réalisée grâce à l'algorithme à clef publique RSA (Rivest, Shamir et Adelman).
- L'authentification, qui sera, elle aussi assurée par l'algorithme RSA.

Ce cryptosystème, illustré par la figure I.3.13, aura besoin en plus de toutes les fonctions cryptographiques :

- D'un générateur de nombre aléatoire pour la génération des clefs de session secrètes.
- D'une mémoire, bien protégée contre les attaques, pour le stockage de la clef privée de l'utilisateur.

Les clefs publiques doivent être placées dans une banque de clefs accessible par tous les utilisateurs. Ces clefs être signées par une autorité de certification de confiance.
Notre solution reste cependant très évolutive et sa conception modulaire permet des adaptations aisées.

L'objectif assigné, à savoir, concevoir un processeur dédié FPGA a été atteint. Il reste néanmoins évident qu'il constitue une excellente base de départ pour son adaptation (que nous jugeons très aisée) à tout autre algorithme même non cryptographique. La fonction processeur pourra toujours être maintenue.
Des améliorations peuvent aussi être apportées au module processeur par le choix d'une autre classification et d'un autre type de processeur déduit des considérations d'optimisation du jeu d'instructions

Parmi les multiples domaines d'applications du cryptoprocesseur, nous pouvons citer la protection de la mémoire vive d'un ordinateur. Cette dernière reste l'endroit le plus sujet à des attaques. Pour ce prémunir contre ces attaques, il serait très intéressant de chiffrer toutes les informations qui s'y trouvent. Nous pouvons donc envisager le couplage d'un microprocesseur généraliste avec un cryptoprocesseur travaillant à son insu à la manière d'un contrôleur de mémoire cache.

## Bibliographie

[1] SCHNEIER B.- Cryptographie Appliquée .- Ed.corr. et trad .- Paris : International Thomson Publishing France, 1995 .- ISBN : 2-84180-000-8

[2] STINSON D.- Cryptographie : Théorie et Pratique .-Paris : International Thomson Publishing France,1996 .- ISBN : 2-84180-013-X

[3] J. Goubert F. Hoornaert and Y. Desmedt. Efficient hardware implementation of the DES. In G.R. Blakley and D. Chaum, editors, Advances in Cryptology : Proceedings of CRYPTO'84, number 196 in Lecture Notes in Computer Science, pages 147-173, Berlin, Germany, 1985. International Association for Cryptologic Research, Springer-Verlag.

[4] A. Matusevich R.C. Fairfield and J. Plany. An LSI digital encryption processor. In In G.R. Blakley and D. Chaum, editors, Advances in Cryptology : Proceedings of CRYPTO'84, number 196 in Lecture Notes in Computer Science, pages 115-143, Berlin, Germany, 1985. International Association for Cryptologic Research, Springer-Verlag.

[5] J. Vandewalle I. Verbauwhede, F. Hoornaert and H.J. De Man. Security and performance optimization of a new DES data encryption chip. IEEE Journal of Solid-State Circuits, 23(3):647-656, June 1988.

[6] A.G. Broscius and J.M. Smith. Exploiting parallelism in hardware implementation of the DES. IN J. Feigenbaum, editor, Advances in Cryptology - CRYPTO '91. Proceedings, number 576 in lecture Notes in Computer Science, pages 367-376, Berlin, Germany, 1992. Int. Assoc. Cryptologic Res, Springer-Verlag.

[7] H. Eberle and C.P. Thacker. A 1 Gbit/second GaAs DES chip. In Proceedings of the IEEE 1992 Custom Integrated Circuits Conference, pages 19.7/1- 4, New York, NY, USA, 1992. IEEE, IEEE.

[8] H. Eberle. A high-speed DES implementation for network applications. In E.F. Brickell, editor, Advances in Cryptology - CRYPTO '92. 12th Anual International Cryptology Conference Proceddings, Lecture Notes in Computer Science, pages 512 - 539, Berlin, Germany, 1993. Springer-Verlag.

[9] KAPS J. - P., PAAR ch .- Fast DES Implementations FPGAs and its Application to a Universal Key-Search Machine.- Worcester Polytechnic Institute.

[10] PATTERSON C.- High Performance DES Encryption in Virtex™ FPGAs using JBits™ .- Colorado : XILINX, Inc.

[11] XILINX Corporation.- Data Book, 1997.

[12] OC-48 Triple-DES : *10x* inc., 1999-2000. http://www.10x.com.

[13] Põldre J.- CRYPTOPROCESSOR PLD001. - Thèse, Department of Computer science, Tallinn Technical University, 1998. - Master. - G.électrique.

[14] ALLIANCE CORE.- DES Cryptoprocessor.- New York : CAST inc., Janvier 2000.

[15] Algorithmic and Register-Transfer Level Synthesis,.../D.E. thomas, E.D. Lagnese, R.A. walker,...[et al.] .- Boston : Kluwer Publishers,1990 .- ISBN : 0-7923-9053-9

[16] DUTRIEUX L., DEMIGNY D.- Logique programmable .-Paris : Eyrolles, 1997 .- ISBN : 2-212-09581-3

[17] VHDL : Langage, Modélisation, Synthèse / R. Airiau, J.M. Bergé, V. olive,...[et al.] .-2 éd. rev. et augm .- Lausanne : Pressses Polytechniques et Universitaires Romandes,1998 .- ISBN : 2-88074-361-3

[18] WEBER J., MEAUDRE M.- VHDL : du langage au circuit,...- Paris : Masson, 1997 .- ISBN : 2-225-82957-8 + CD-ROM.

[19] BLOTIN T.- Le Langage de Description VHDL.- Lycée Paul Eluard Saint-Denis.

[20] REESE B.- Logic Synthesis with VHDL : Combinational Logic.- Mississipi State University (Electrical Engineering Departement).

[21] REESE B.- Logic Synthesis with VHDL : Sequential Circuits.- Mississipi State University (Electrical Engineering Departement).

[22] REESE B.- Logic Synthesis with VHDL : System Synthesis.- Mississipi State University (Electrical Engineering Departement).

[23] DAVIO M., DESCHAMPS J.-P., THAYSE A.- Machines algorithmiques. -Lausanne : Presses Polytechniques Romandes, [s.d.] .- ISBN : 2-88074-015-0

[24] GONTENSOU J.-N.- Le câblage des algorithmes .- Paris : Hermes, 1995 .- ISBN : 2-86601-469-3

[25] BAUGE M., BONIFAS D.- Conception des circuits à très haute intégration .- Techniques de l'ingénieur, traité informatique H 690.

[26] VAN DEN BOUT D .- The Pratical XILINX ® Designer LAB Book Version 1.5.- New Jersey : Prentice Hall, 1999 .- ISBN : 0-13-021617-8

[27] BETH T., FRISCH M. et SIMMONS G. J.- Public Key Cryptography : State of the Art and Future Directions, Lecture Notes in Computer Science 578. Spinger-Verlag, Berlin, 1992.
[28] TUAL J.-P.- Cryptographie.- Techniques de l'Ingénieur, H2248.

[29] SAPORTA, G.- Théorie et méthodes de la statistique. - Paris : Techniq, 1978.- 376p.

[30] DIFFIE W.- The First Ten Years of Public-Key Cryptography.- Proceedings of the IEEE, vol. 76, n° 5, p. 560-577, mai 1988.

[31] DIFFIE W.- The First Ten Years of Public-Key Cryptography.- SIMMONS G. J.- Contemporary Cryptology : The Science of Information Integrity, p. 65-134. IEEE Press, Piscatoway, NJ, 1992.

[32] NEEDHAM R. M., SCHROEDER M. D.- Using Encryption for Authentification in Large Networks of Computers. Communications of the ACM, vol. 21, n° 12, p. 993-999, décembre 1978.

[33] HASKINS G.M.- Securing Asynchronous Transfer Mode Networks .- Thèse , Worcester Polytechnic Institute, 1997 .- Master.- G.Electrique.

[34] MEINADIER J.- P.- Structure et fonctionnement des ordinateurs .- Paris : Librairie Larousse, 1975 .- ISBN : 2-03-070359-1

[35] TANENBAUM A.- Architecture de l'ordinateur : du circuit logique au logiciel de base .- Paris : InterEditions, 1987 .- ISBN : 2-7296-0133-3

[36] KAPS J.-P.- High Speed FPGA Architectures for the Data Encryption Standard. - Thèse , Worcester Polytechnic Institute, 1998. – Master .- G.Electrique.

# Annexe A

Nous consacrerons cet annexe aux modèles physiques obtenus par l'implémentation de l'algorithme DES, de ses différentes options architecturales et du cryptoprocesseur sur le circuit FPGA XC4013XL-3BG256. Tout ces modèles physiques sont fournis par l'outil FLOORPLANNER du logiciel XILINX Foundation Series 1.5, Student Version.

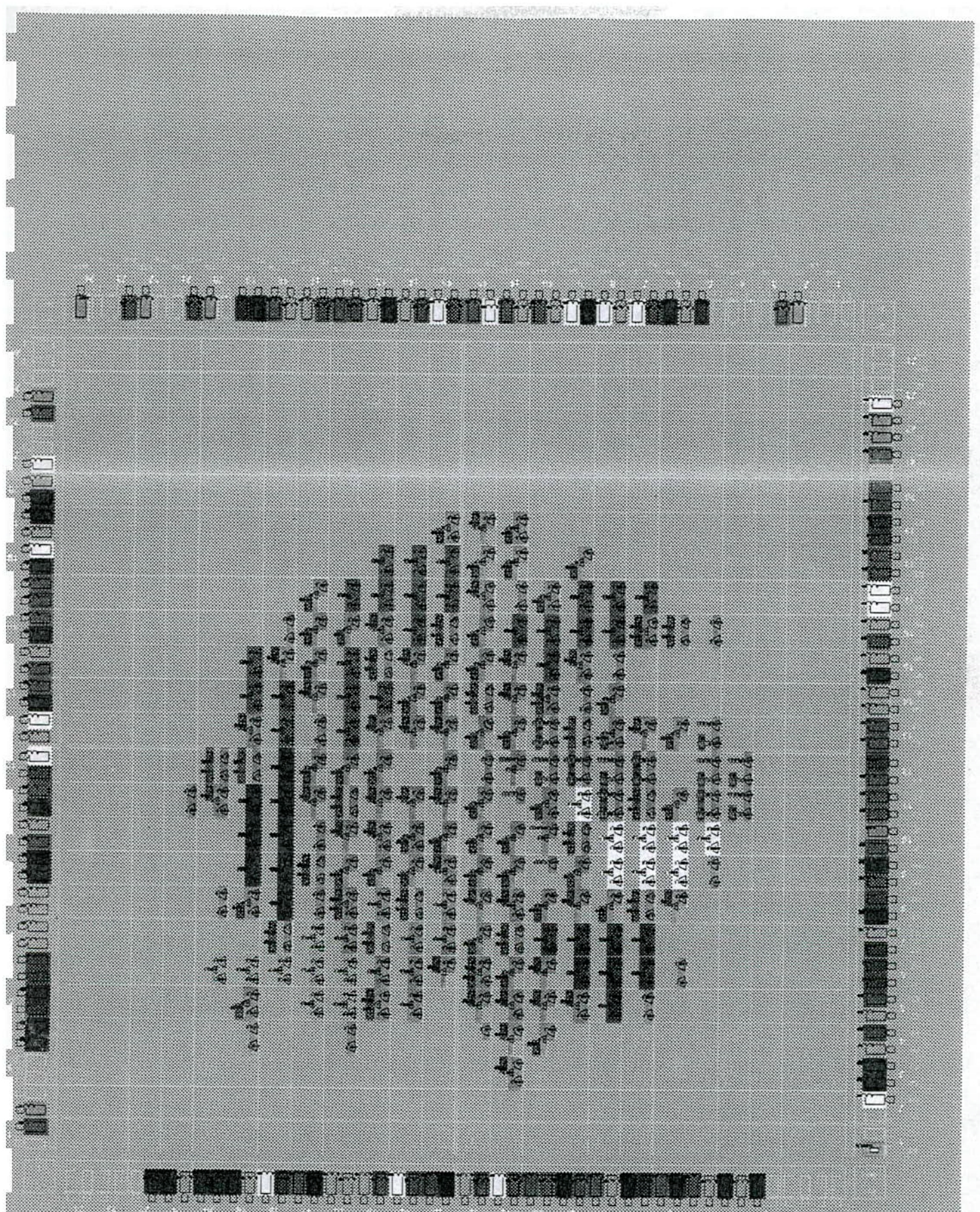


**Figure A.1 : Modèle physique de l'implémentation de l'algorithme DES, avec une structure de base (DES_VER1), sur le circuit FPGA XC4013XL-3BG256**

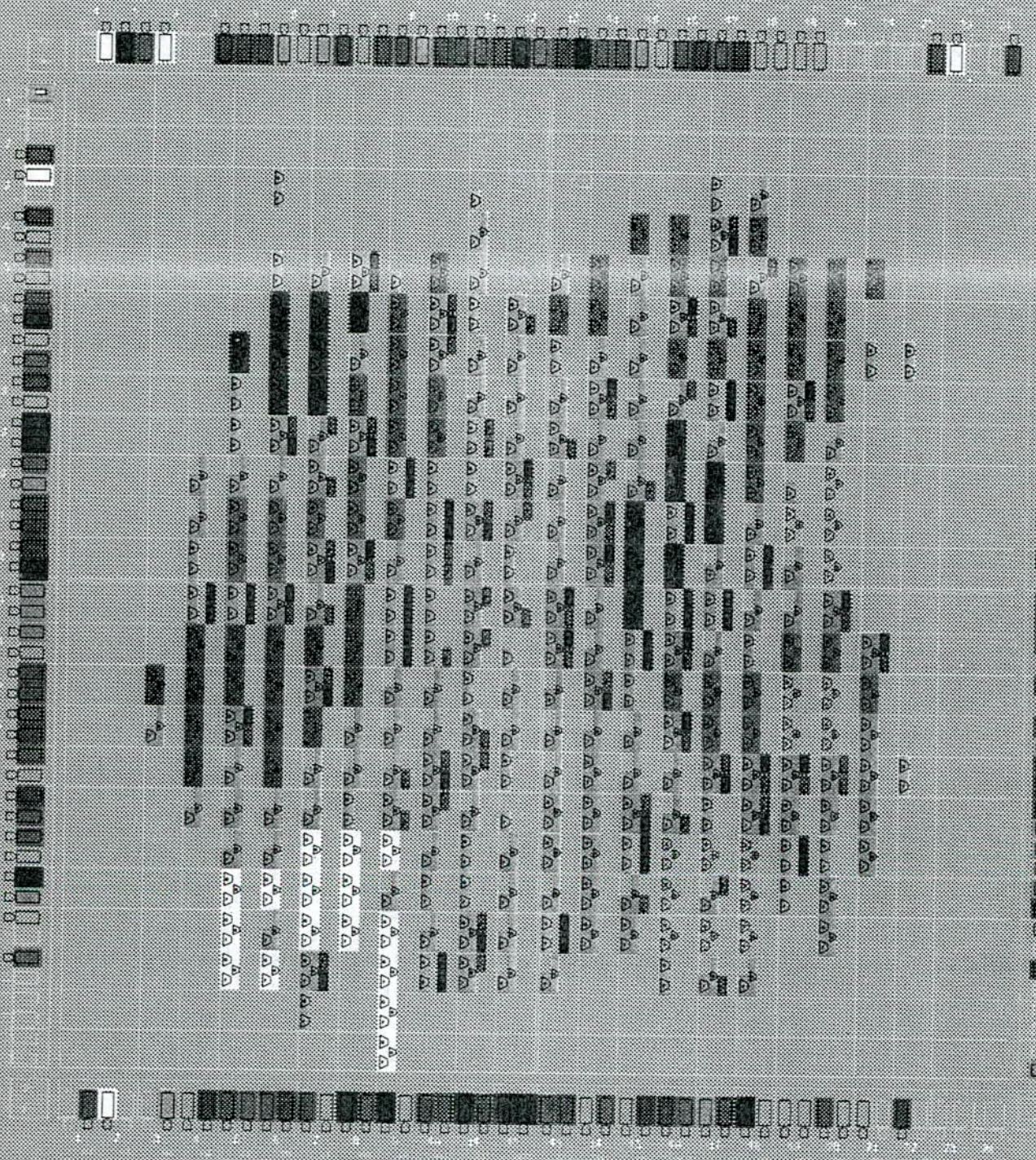


**Figure A.2 : Modèle physique de l'implémentation de l'algorithme DES, avec une structure en boucle expansée deux fois (DES_VER2.1), sur le circuit FPGA XC4013XL-3BG256**

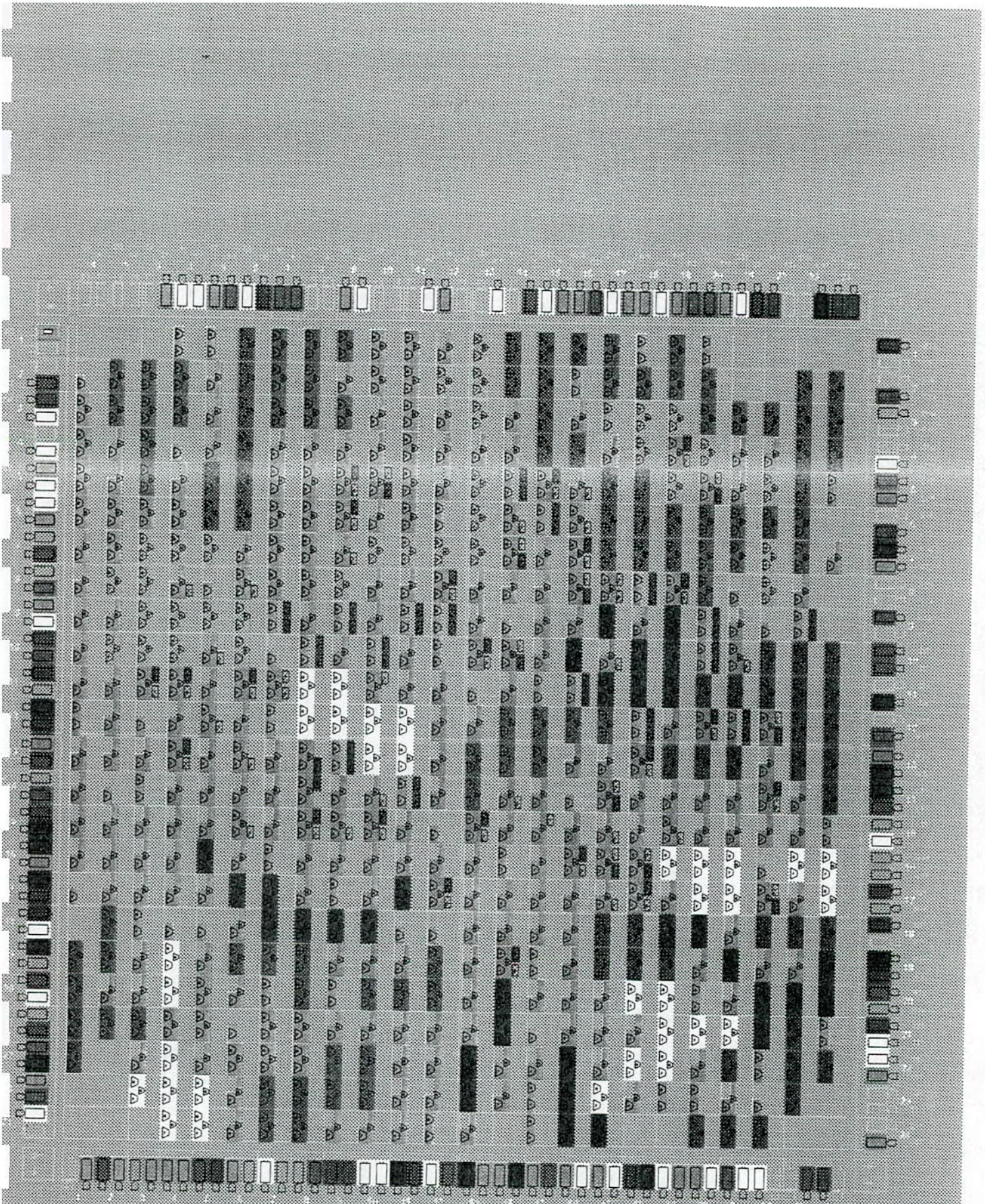


**Figure A.3 : Modèle physique de l'implémentation de l'algorithme DES, avec une structure en boucle expansée quatre fois (DES_VER2.2), sur le circuit FPGA XC4013XL-3BG256**

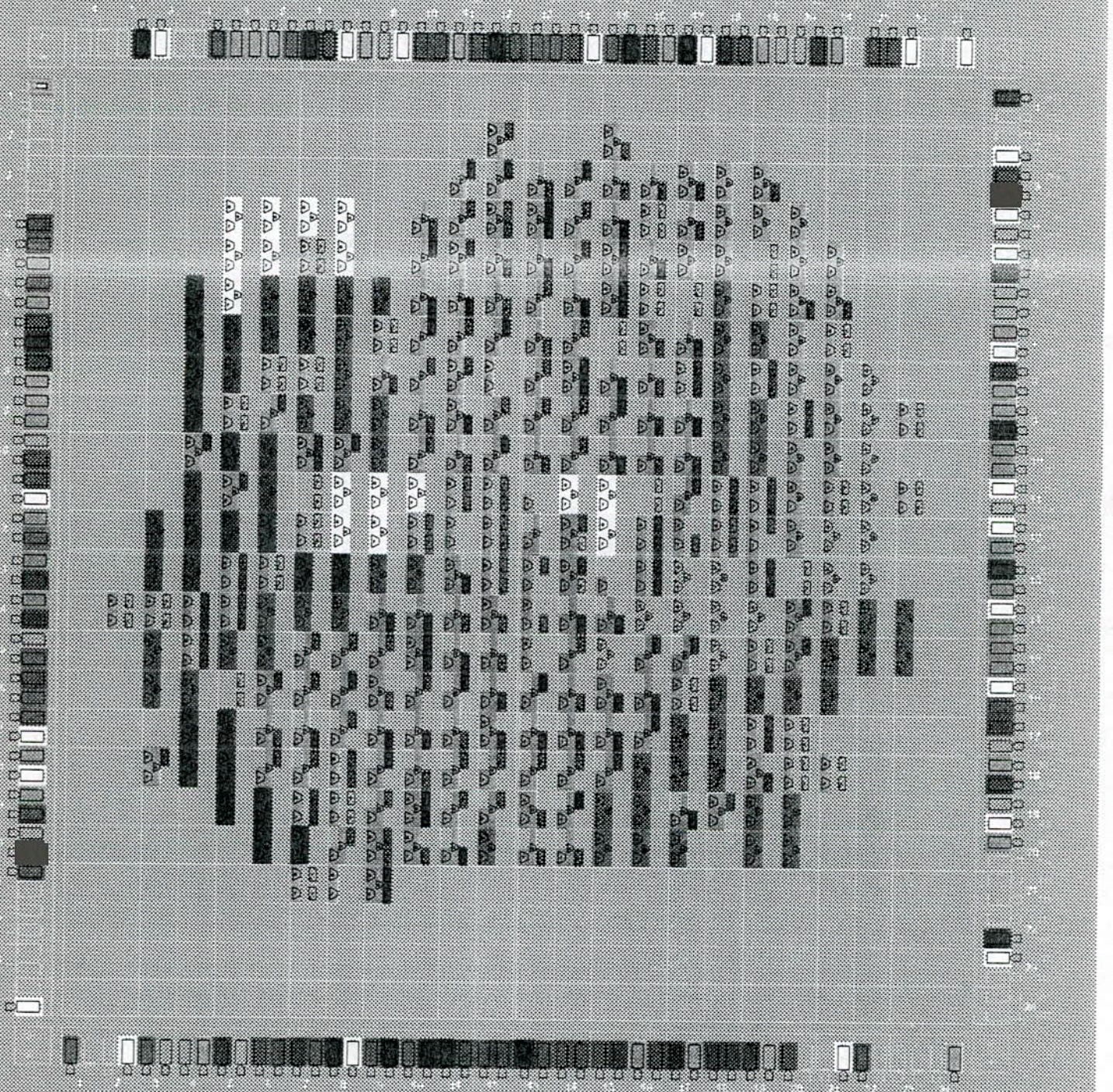


**Figure A.4 : Modèle physique de l'implémentation de l'algorithme DES, avec une structure en pipeline à deux étages (DES_VER3.1), sur le circuit FPGA XC4013XL-3BG256**

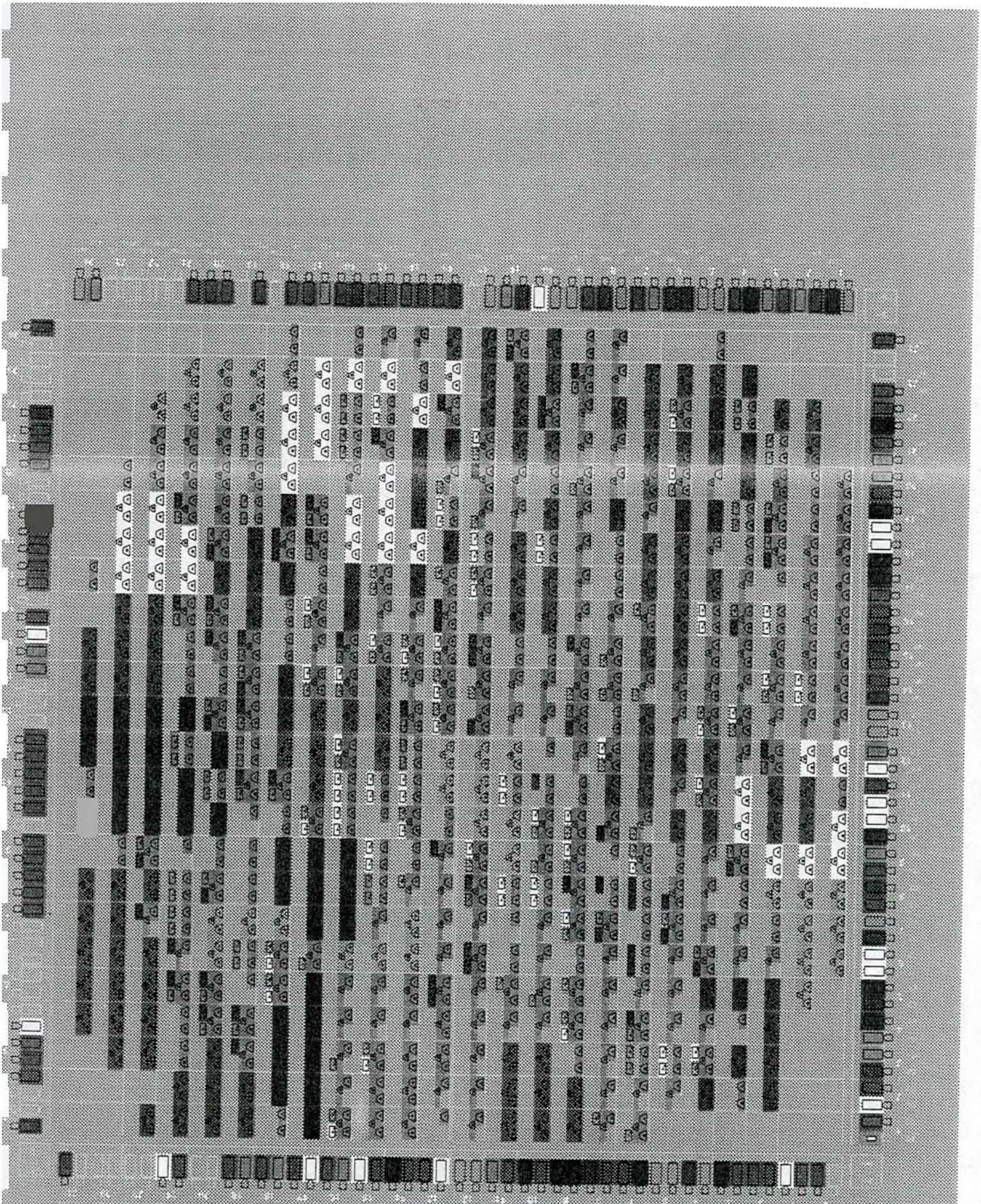


Figure A.5 : Modèle physique de l'implémentation de l'algorithme DES, avec une structure combinée (DES_VER4), sur le circuit FPGA XC4013XL-3BG256

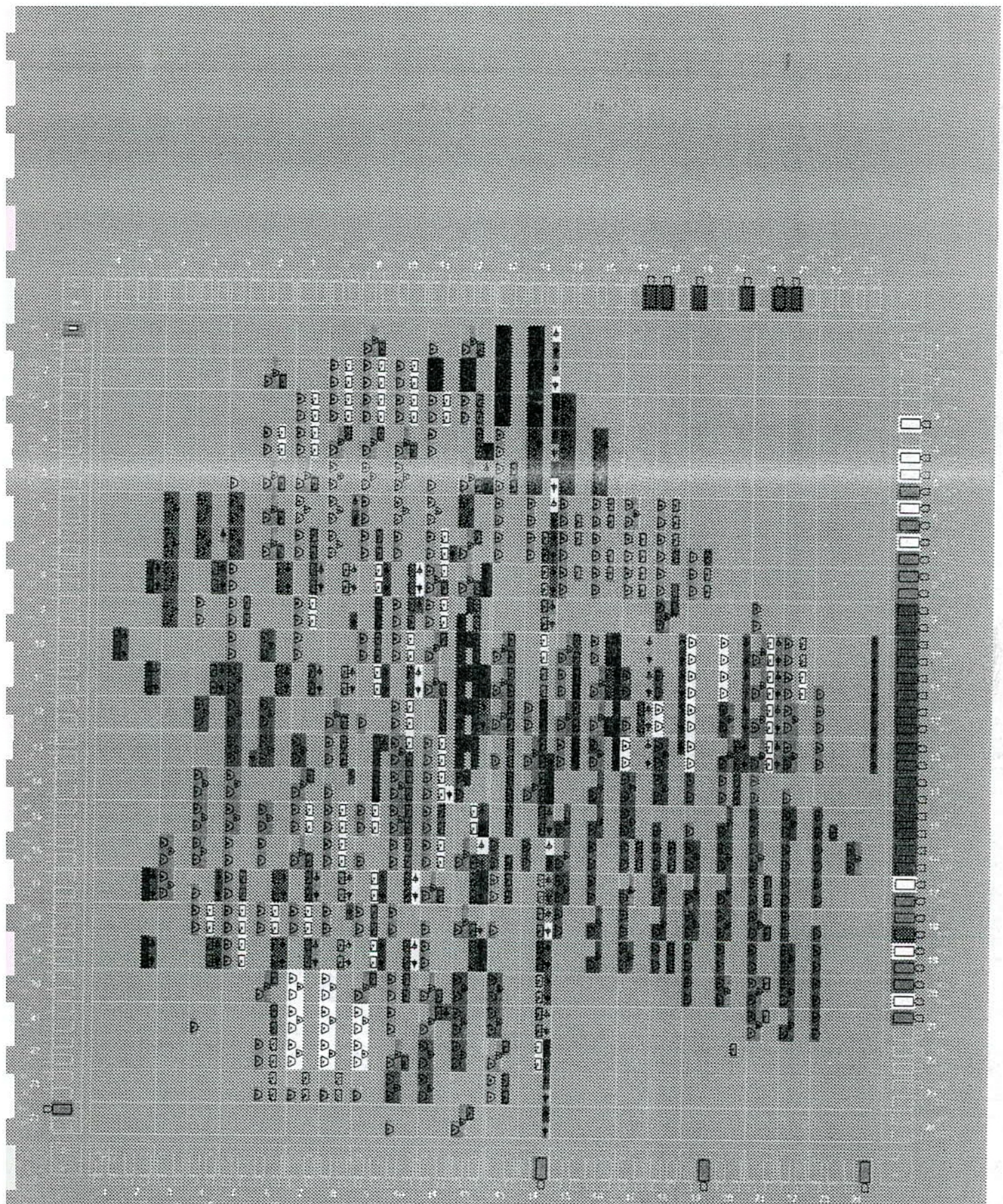


**Figure A.6 : Modèle physique de l'implémentation du cryptoprocesseur sur le circuit FPGA XC4013XL-3BG256**

# Annexe B

Cet annexe englobe les diagrammes temporels des différentes implémentations. Chaque sous-annexe donne le diagramme temporel d'une option architecturale de l'algorithme DES.

Pour chaque implémentation, la période d'horloge de la simulation est prise très proche de la période d'horloge minimum donnée par l'analyseur temporel.

Les blocs de données et leurs clefs correspondantes sont fournis respectivement par les bus PT1_IN[0:63] et CLEF1_IN[0:63]. Le résultat d'une opération de chiffrement ou de déchiffrement apparaît sur le bus CT[0:63].

Toutes les simulations sont faites en prenant pour texte clair le bloc (hexadécimal) 0123456789ABCDEF et pour la clef le bloc (hexadécimal) 133457799BBCDFF1

# Annexe B.1

La figure B.1 donne le diagramme temporel d'une opération de chiffrement et de déchiffrement de l'implémentation version DES_VER1.1 réalisée à base d'une architecture en structure de base de l'algorithme DES.

La période d'horloge est fixée à 50ns. Durant le premier cycle d'horloge il y a chargement d'un bloc de donnée et de sa clef respective. Le résultat d'une opération de chiffrement ou de déchiffrement apparaît après seize cycles d'horloge. Pour cet exemple de simulation, l'opération de chiffrement du texte clair 0123456789ABCDEF donne en résultat le texte chiffré 85E813540F0AB405. Ce texte chiffré subit à son tour une opération de déchiffrement pour donner le texte clair initial 0123456789ABCDEF.

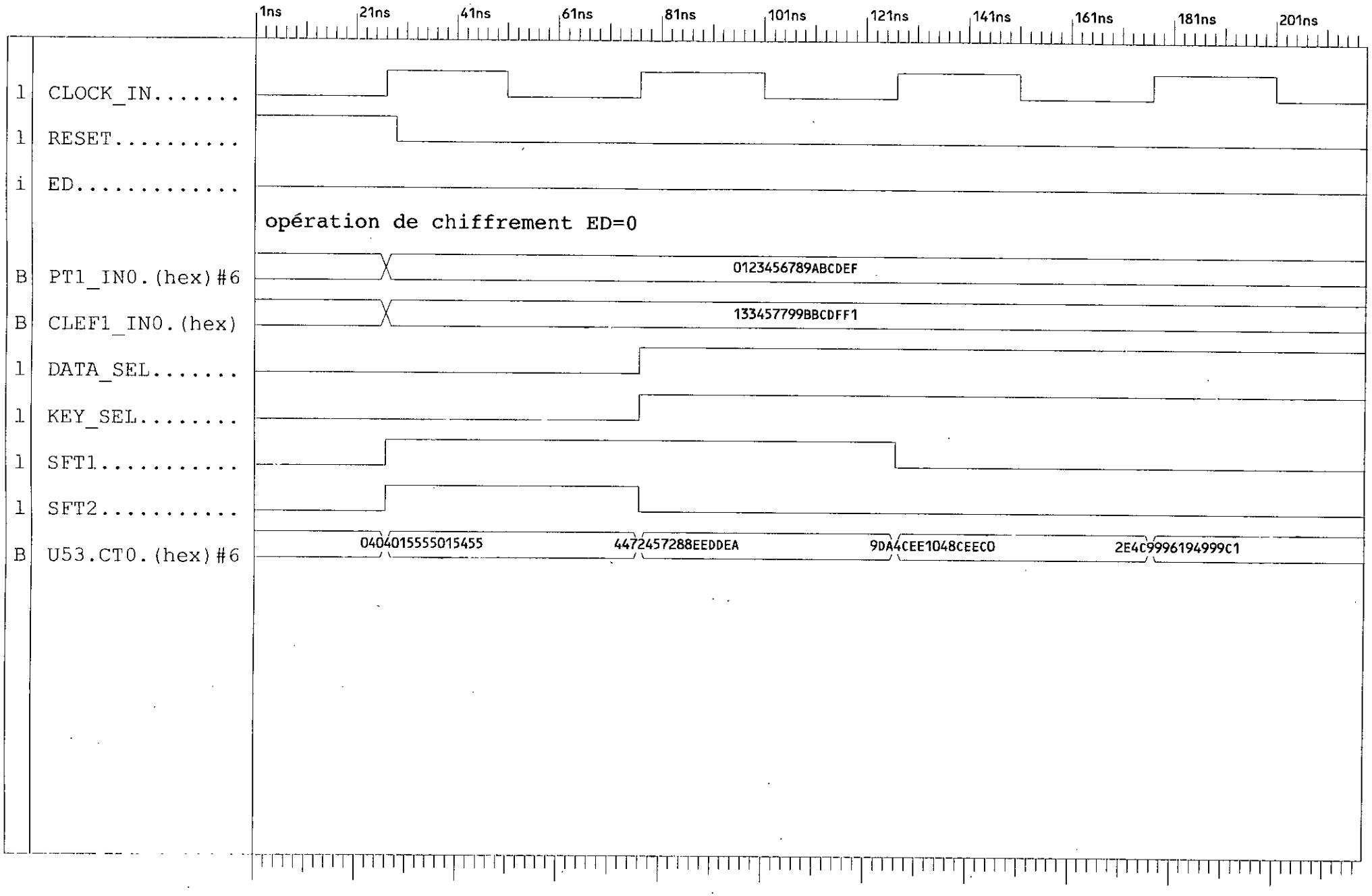



Figure B.1 : Conception DES_VER1.1 (structure de base)

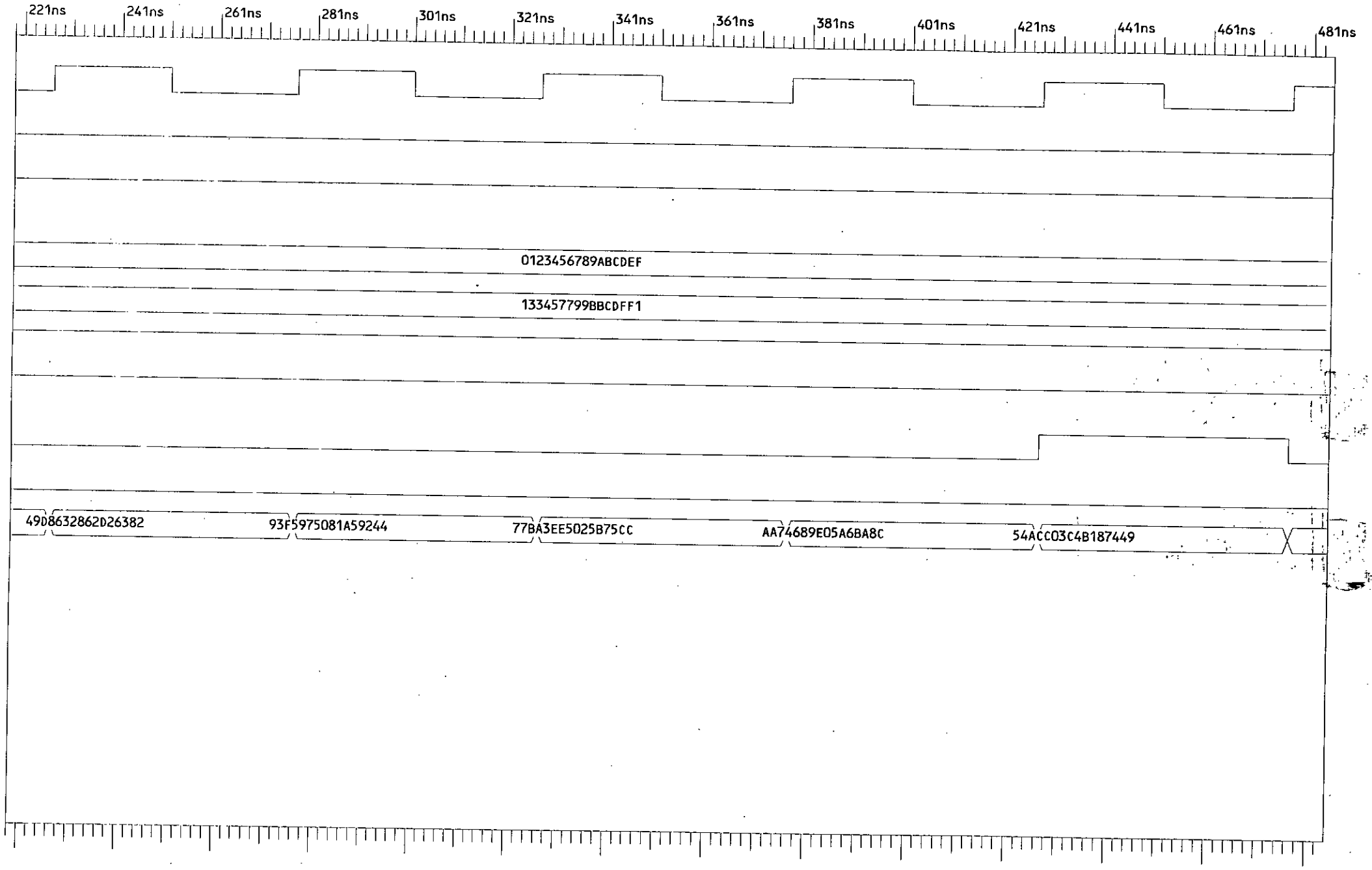



Figure B.1 : Conception DES_VER1.1 (structure de base). suite

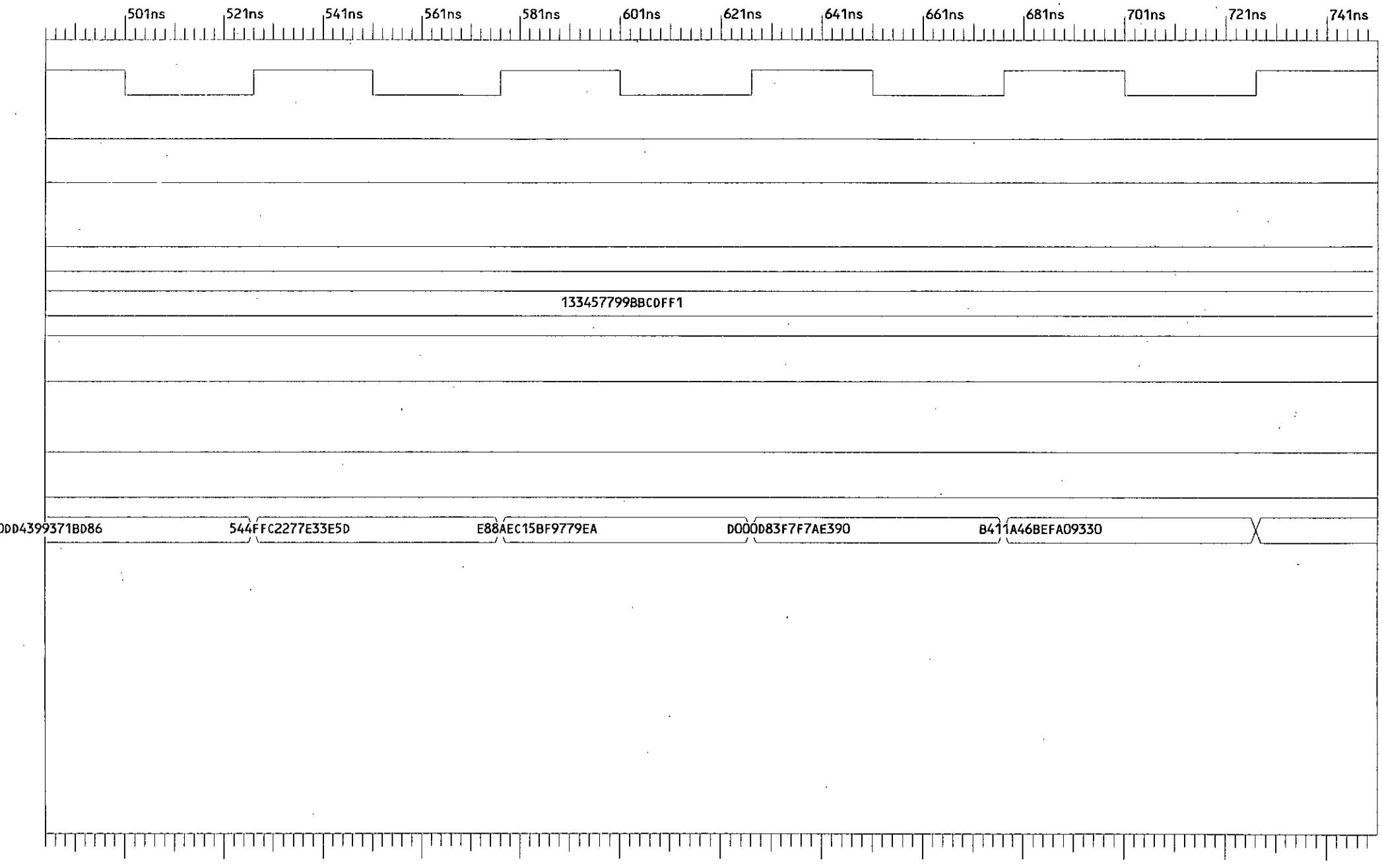



Figure B.1 : Conception DES_VER1.1 (structure de base). suite

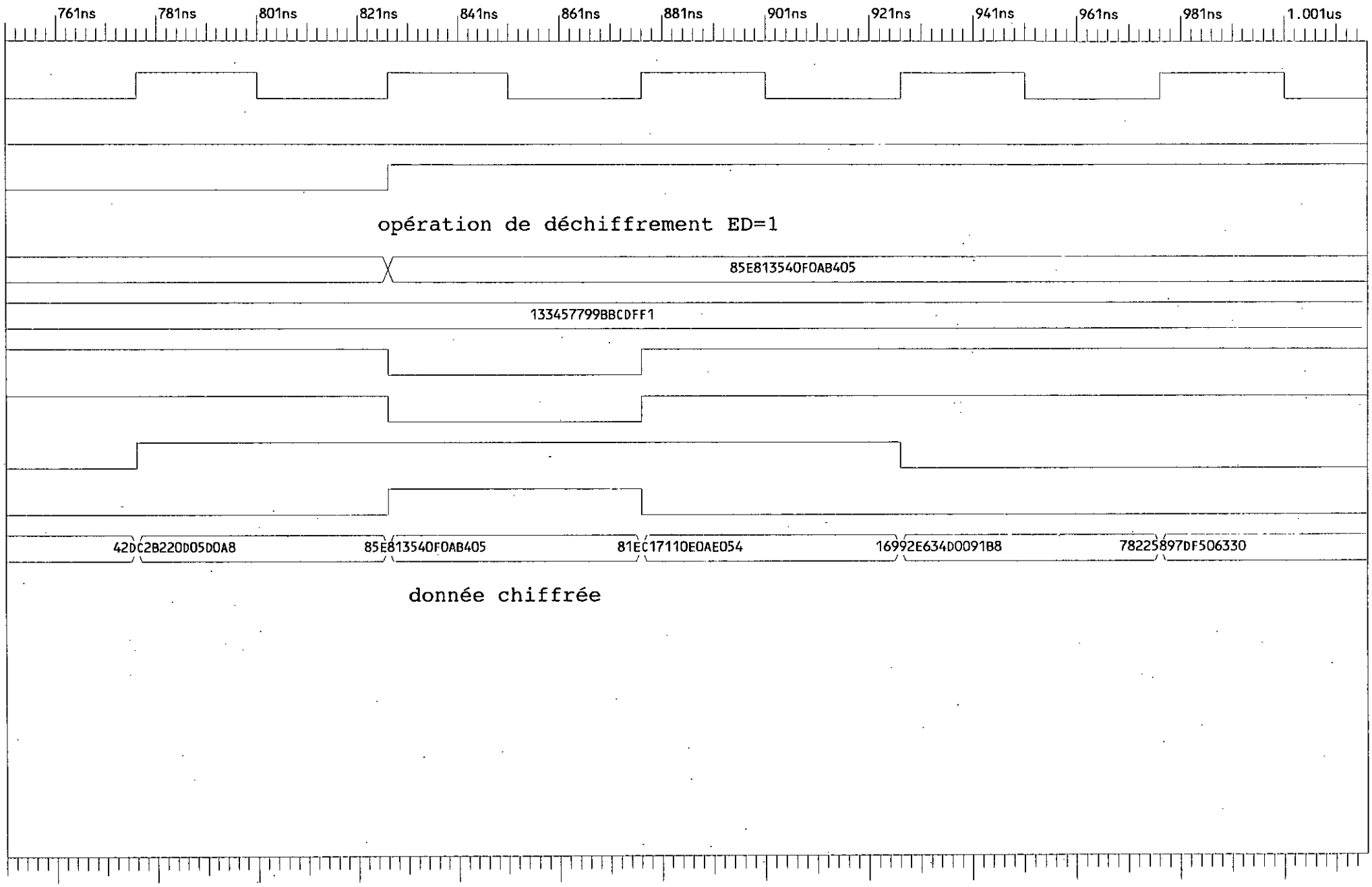



Figure B.1: Conception DES_VER1.1 (structure de base), suite

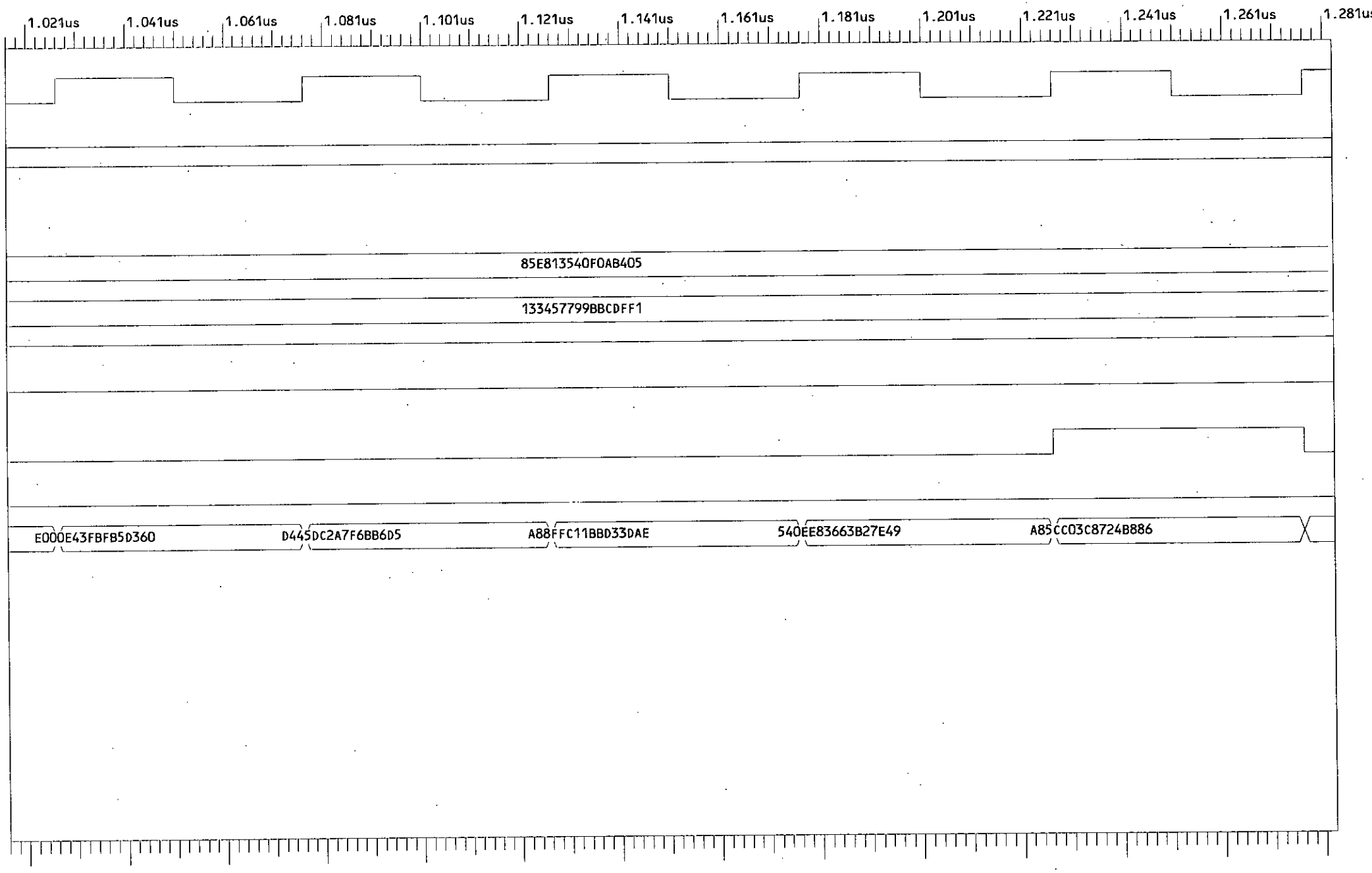



Figure B.1 : Conception DES_VER1.1 (structure de base). suite

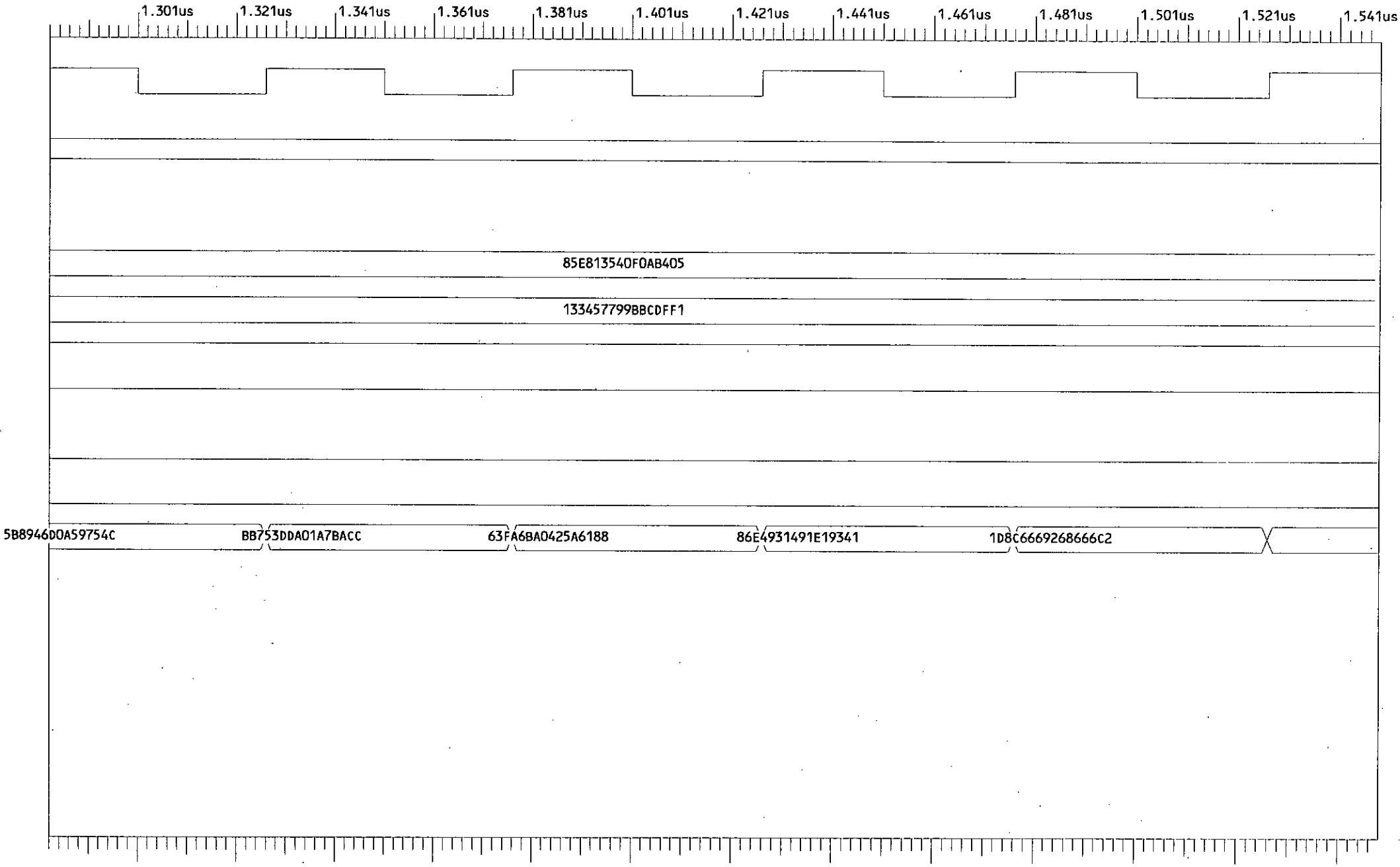



Figure B.1 : Conception DES_VER1.1 (structure de base), suite

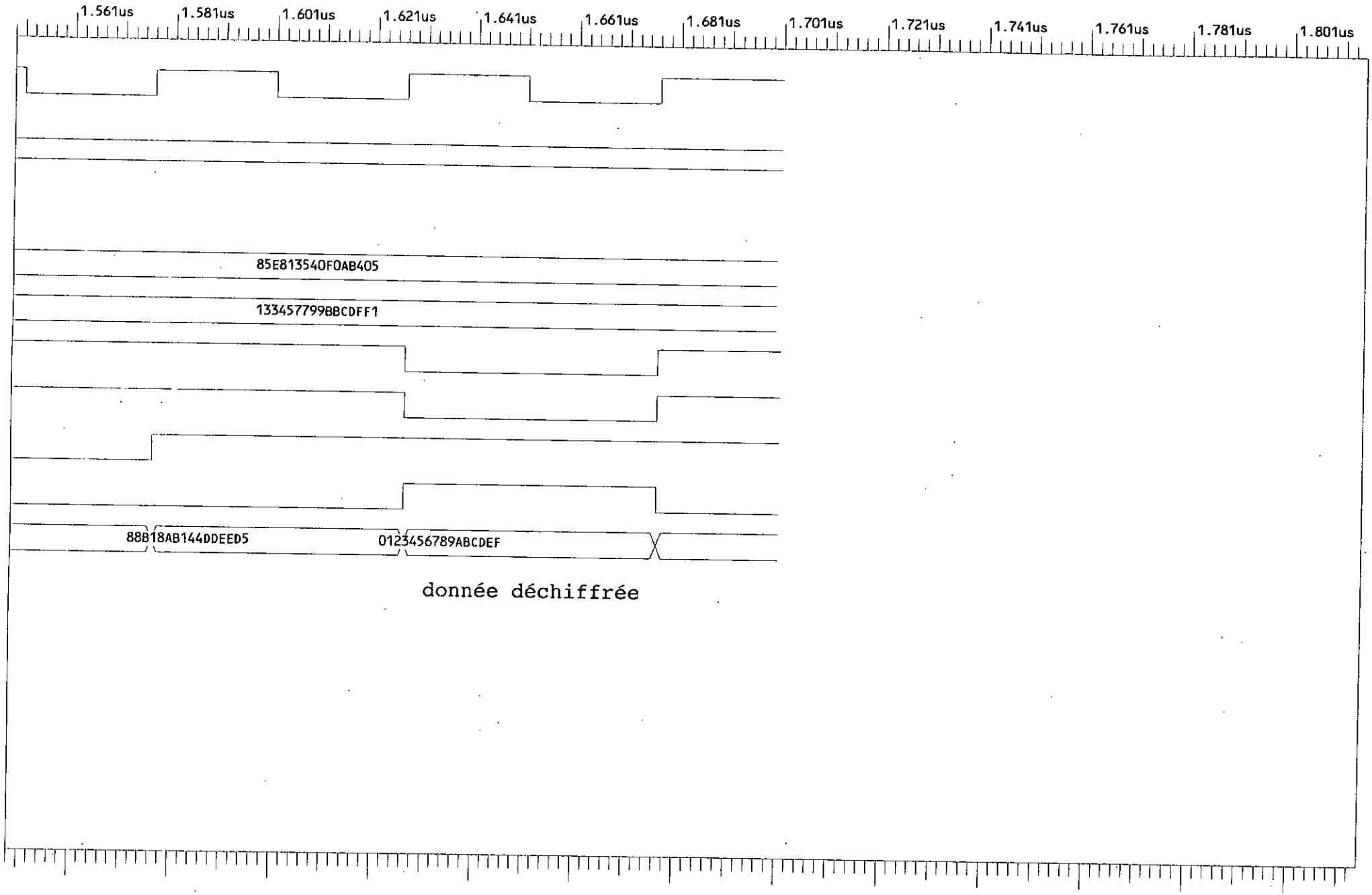



Figure B.1 : Conception DES_VER1.1 (structure de base). suite

## Annexe B.2

La figure B.2 montre le diagramme temporel d'une opération de chiffrement de l'implémentation version DES_VER2.1 qui est à base d'une structure en boucle à deux (02) expansions.

Dans cette simulation la période d'horloge est fixée à 50ns. Pendant le premier cycle d'horloge il y a chargement du bloc de données et de sa clef respective. Le résultat de cette opération de chiffrement apparaît sur le bus CT[0:63] après huit (08) cycles d'horloge.

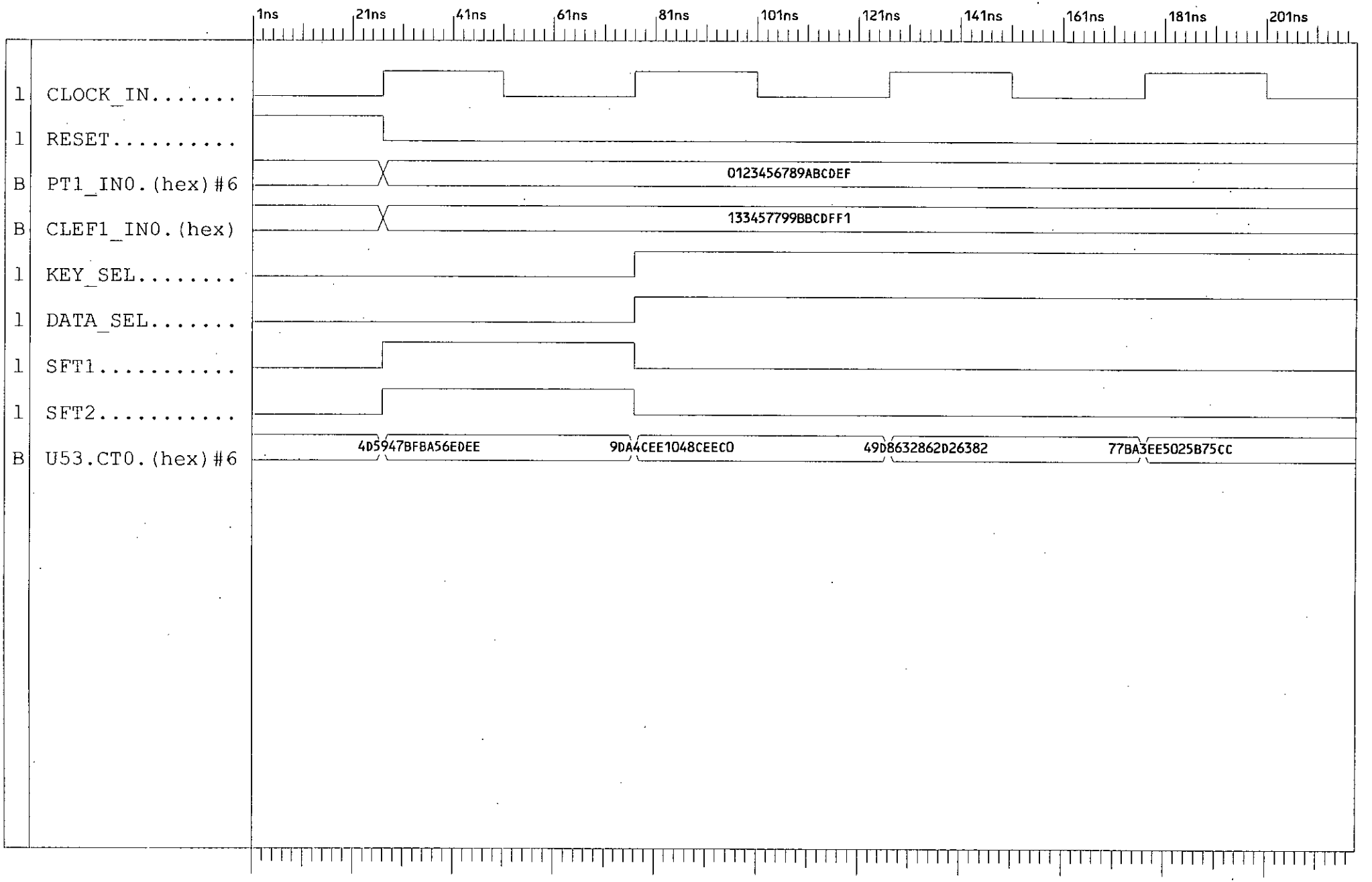



Figure B.2 : Conception DES_VER2.1 (deux expansions de la boucle)

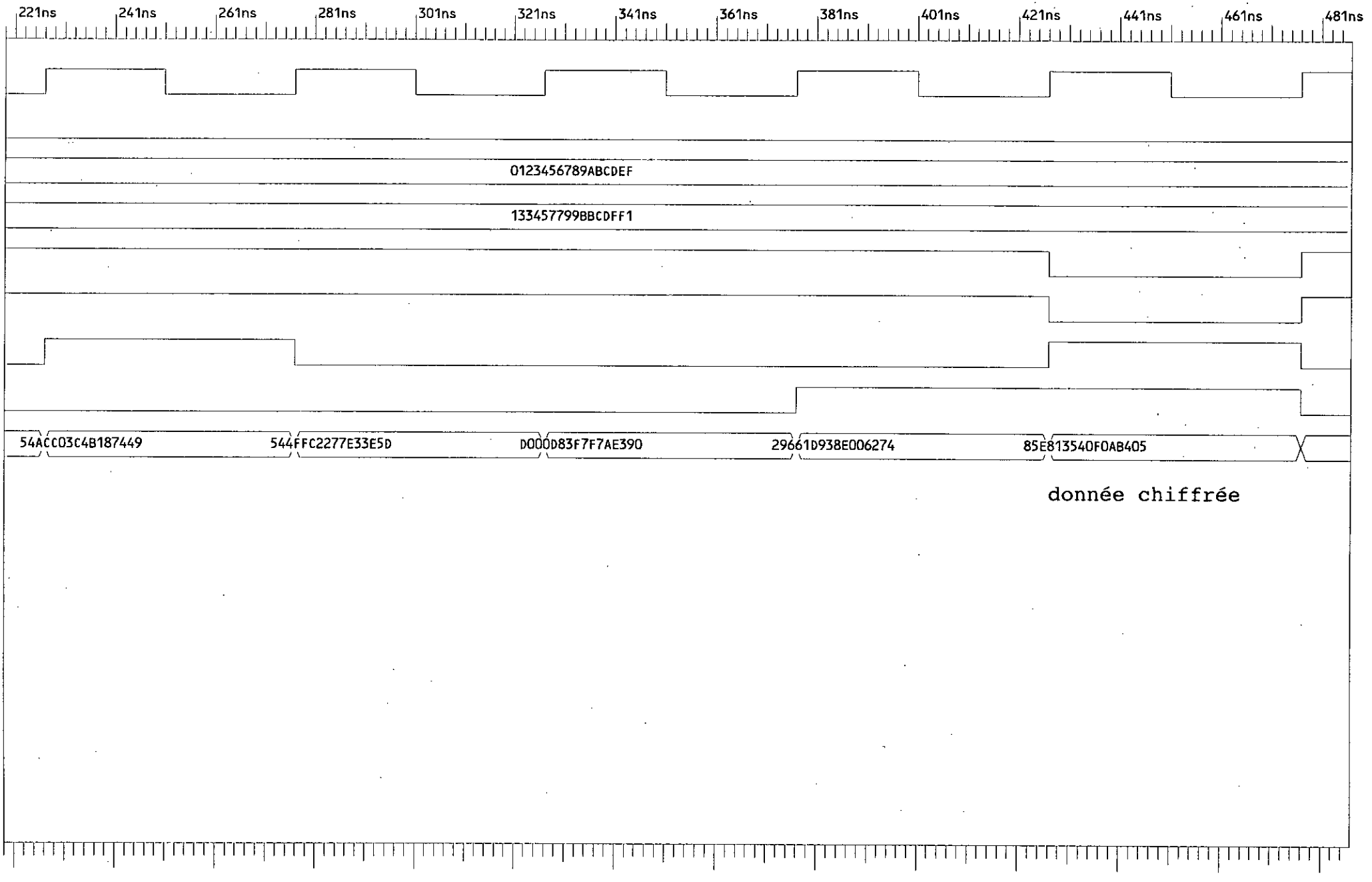



Figure B.2 : Conception DES_VER2.1 (deux expansions de la boucle). suite

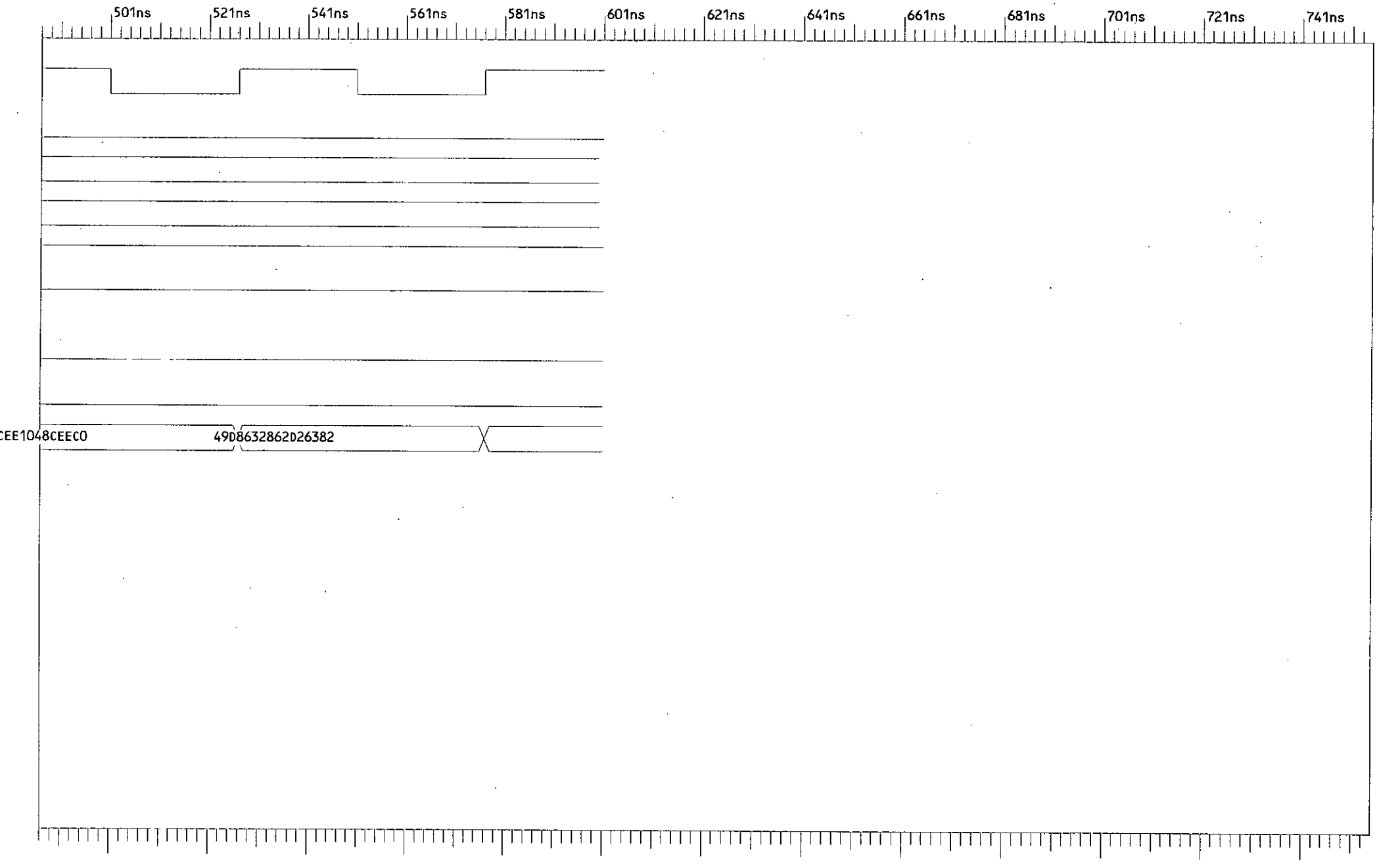



Figure B.2 : Conception DES_VER2.1 (deux expansions de la boucle). suite

## Annexe B.3

Le diagramme temporel de l'implémentation version DES_VER2.2 ayant une structure en boucle à quatre (04) expansions est montré en figure B.3.

La période d'horloge de la simulation est fixée à 100ns. Une paire (donnée, clef) est chargé durant le premier cycle d'horloge. Le bloc chiffré apparaît sur le bus CT[0:63] après quatre (04) cycles d'horloges.

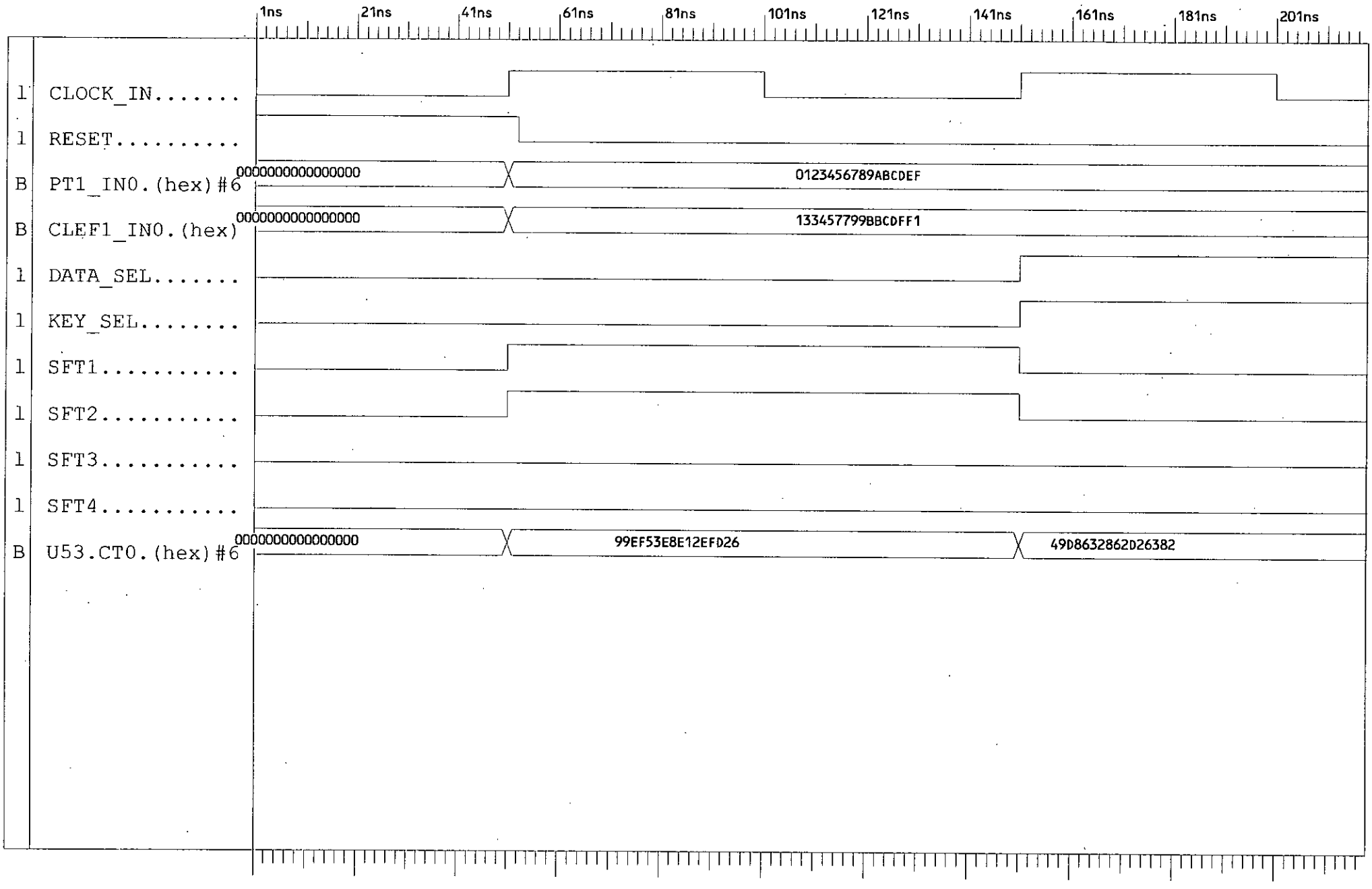



Figure B.3 : Conception DES_VER2.2 (quatre expansions de la boucle)

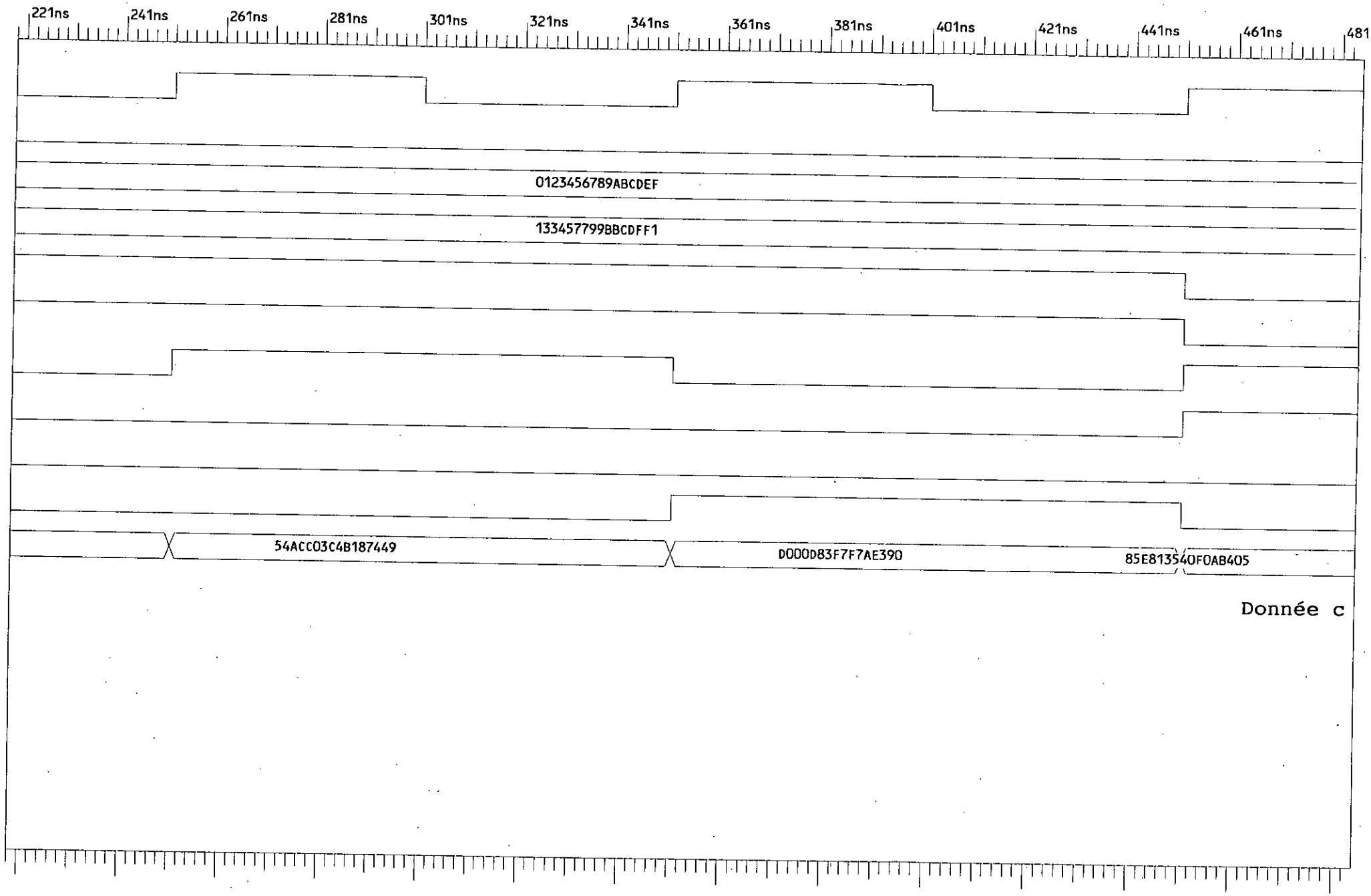



Figure B.3 : Conception DES_VER2.2 (quatre expansions de la boucle), suite

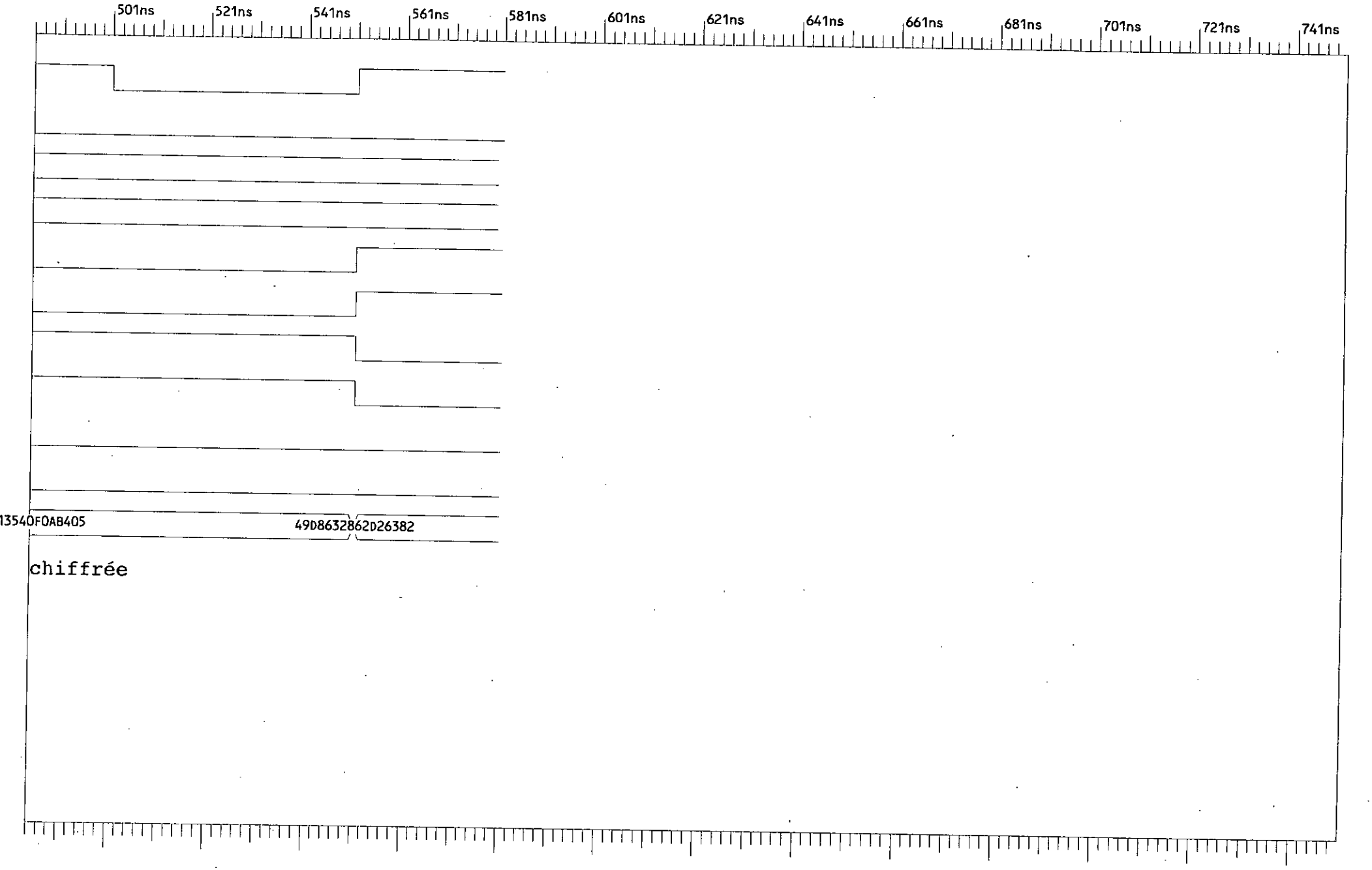



Figure B.3 : Conception DES_VER2.2 (quatre expansions de la boucle). suite

## Annexe B.4

La figure B.4 montre le diagramme temporel de l'implémentation version DES_VER3.1 ayant une structure en pipeline à deux (02) étages.

La période d'horloge est fixée à 50ns. Durant les deux premiers cycles d'horloge il y a remplissage du pipeline par deux blocs de données et leurs clefs respectives. Le résultat du chiffrement est fourni en sortie après seize (16) cycles d'horloge.

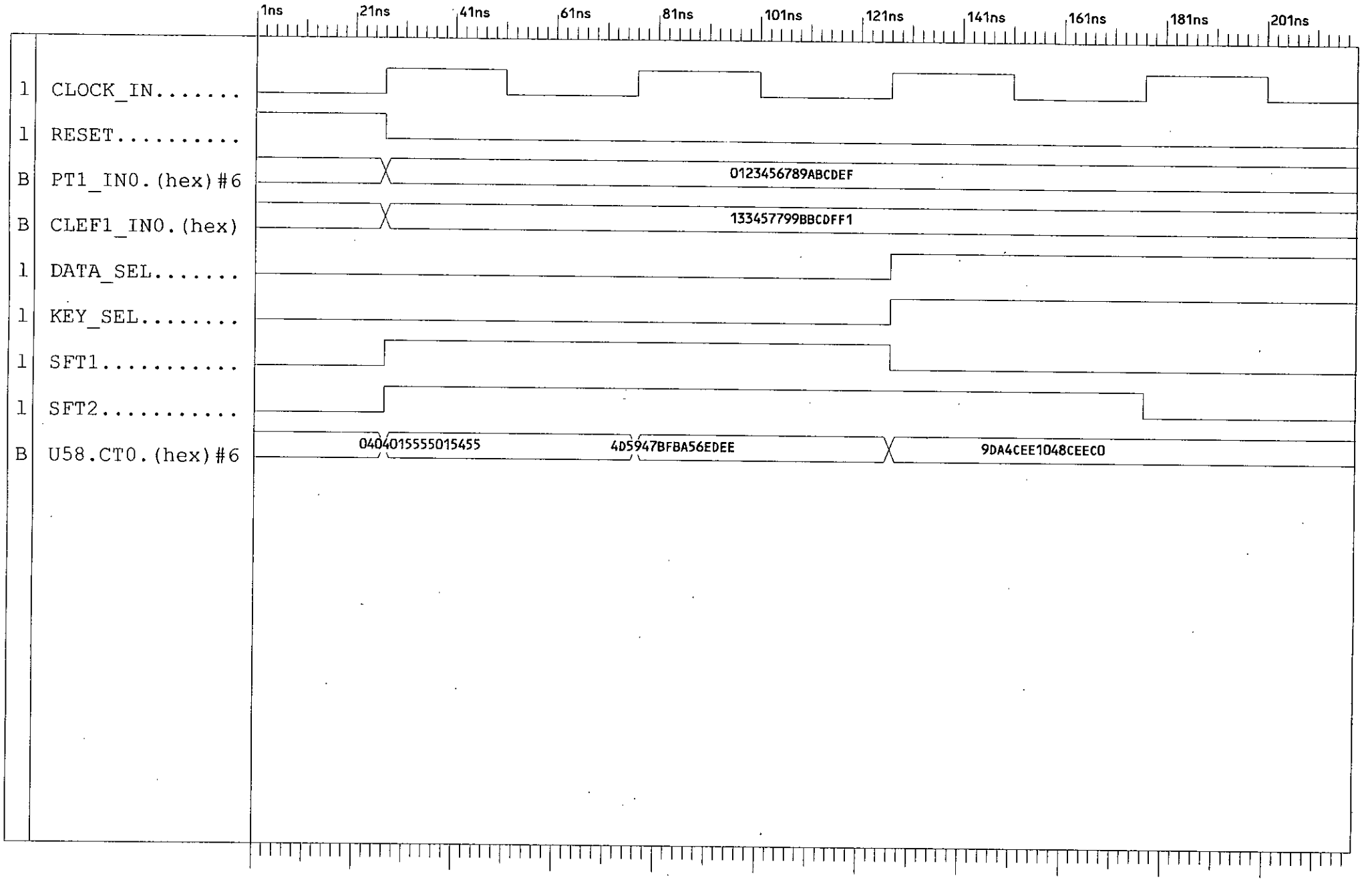



Figure B.4 : Conception DES_VER3.1 (deux pipeline)

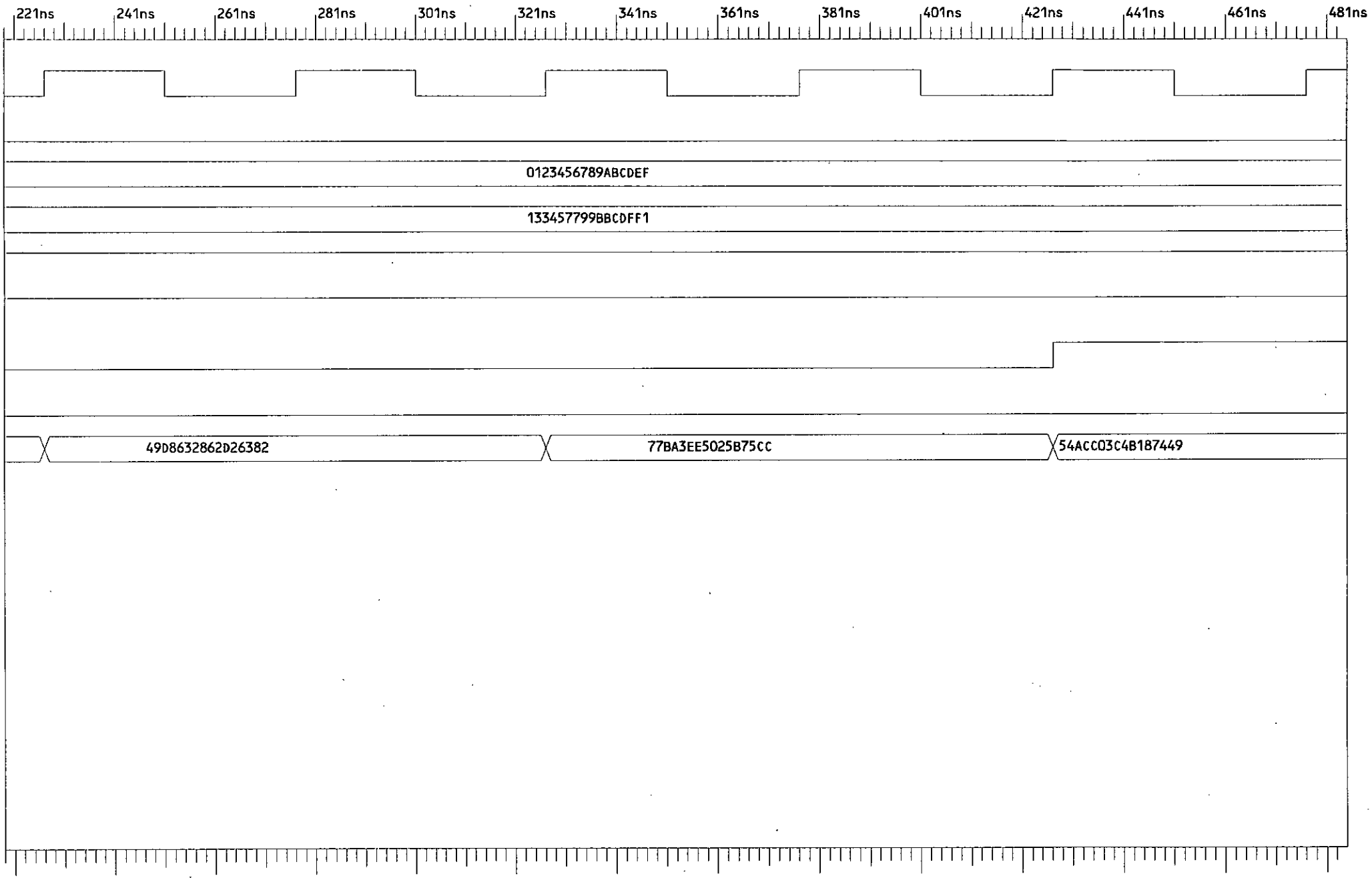



Figure B.4 : Conception DES_VER3.1 (deux pipeline). suite

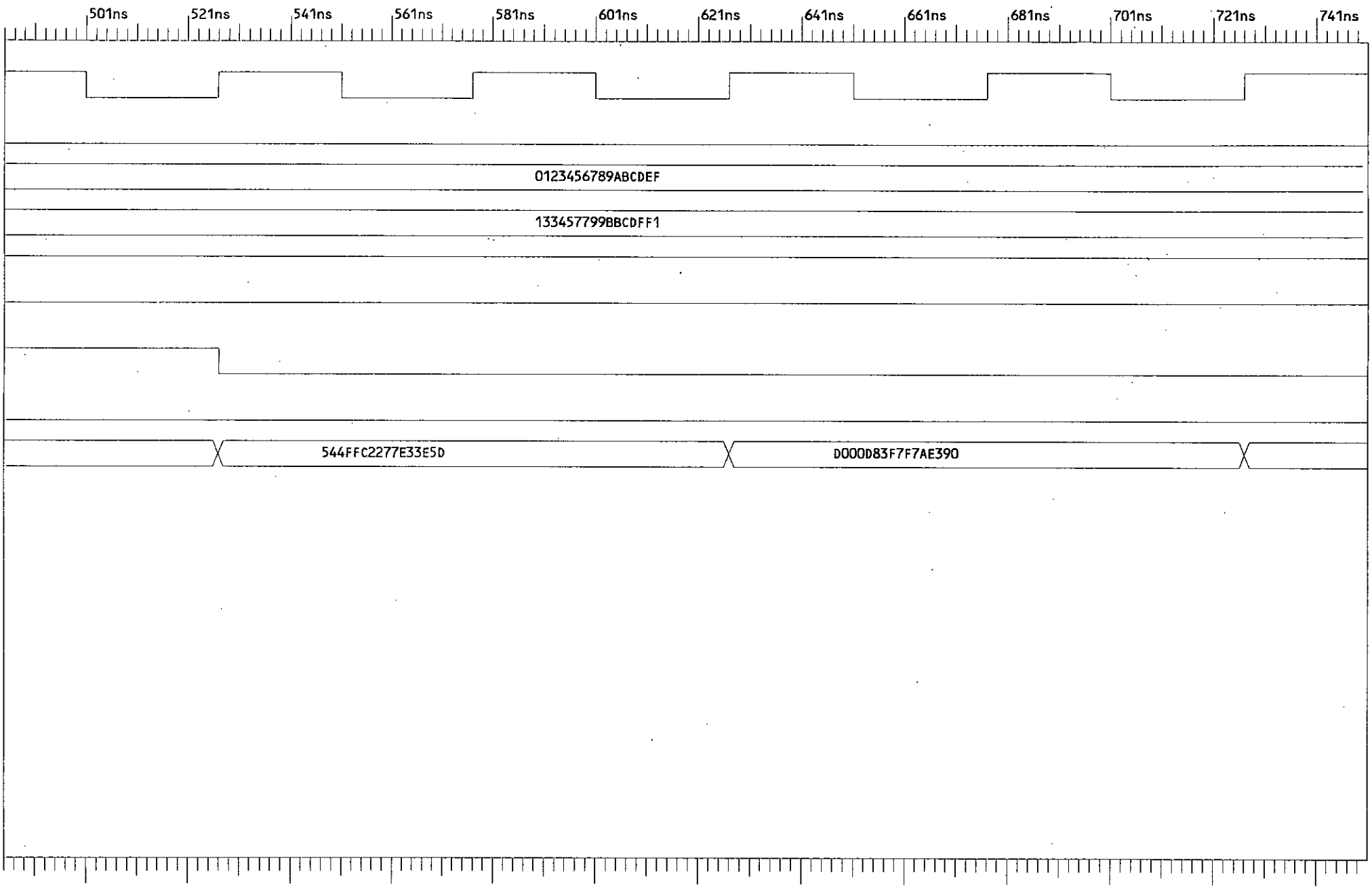



Figure B.4 : Conception DES_VER3.1 (deux pipeline). suite

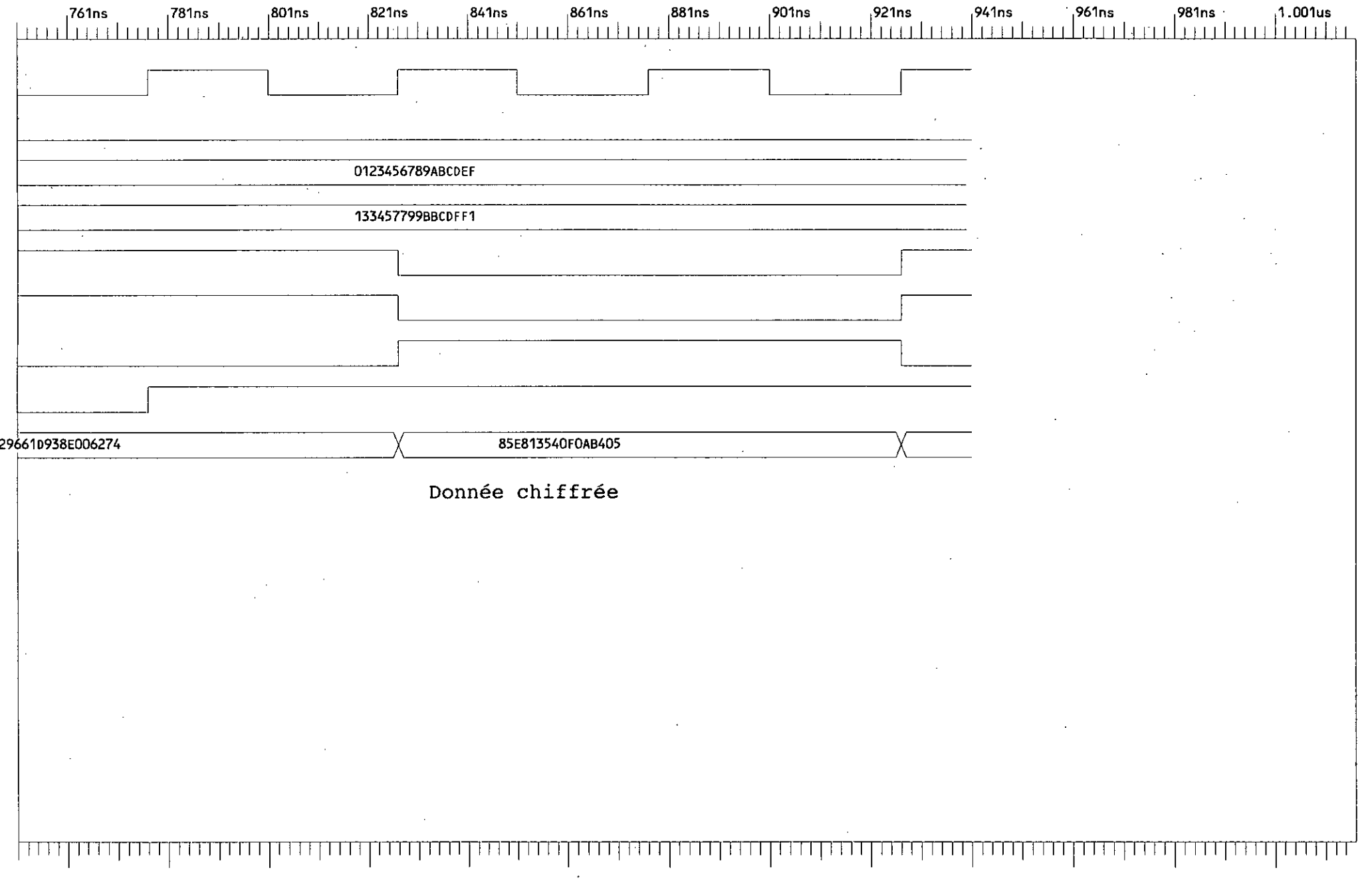



Figure B.4 : Conception DES_VER3.1 (deux pipeline). suite

# Annexe B.5

Le diagramme fonctionnel de la conception DES_VER3.2 ayant structure en pipeline à 4 étages est montré par la figure B.5.

Pendant les 4 premiers cycles d'horloge il y a remplissage du pipeline avec quatre blocs de données de même valeur (0123456789ABCDEF), fournies par l'entrée PT1_IN[0:63], et de leurs clefs respectives (133457799BBCDFF1). Durant chaque cycle d'horloge il y a calcul de quatre blocs de données et leurs clefs respectives. Le résultat apparaît sur le bus CT[0:63] après seize cycles d'horloge.

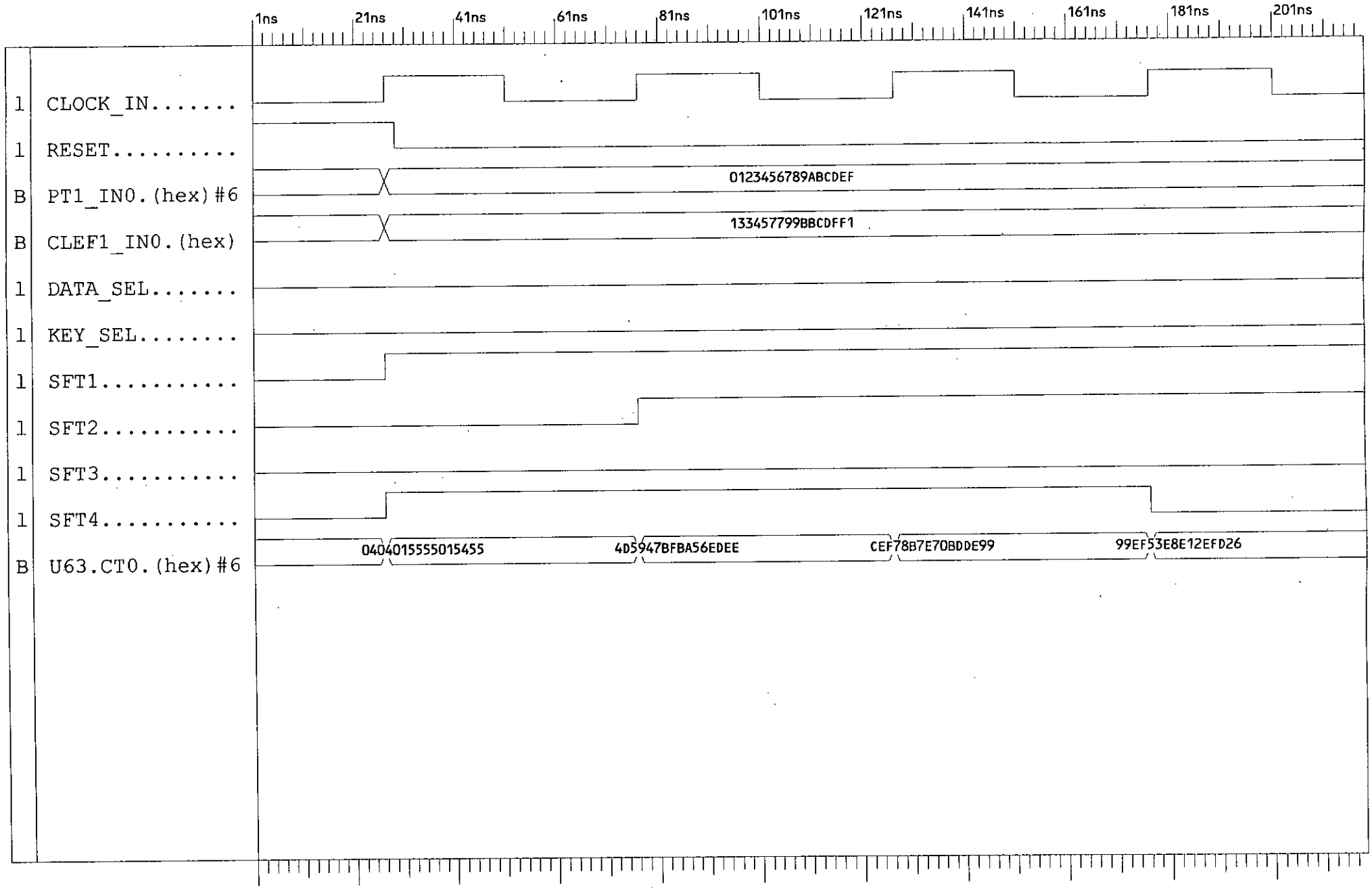
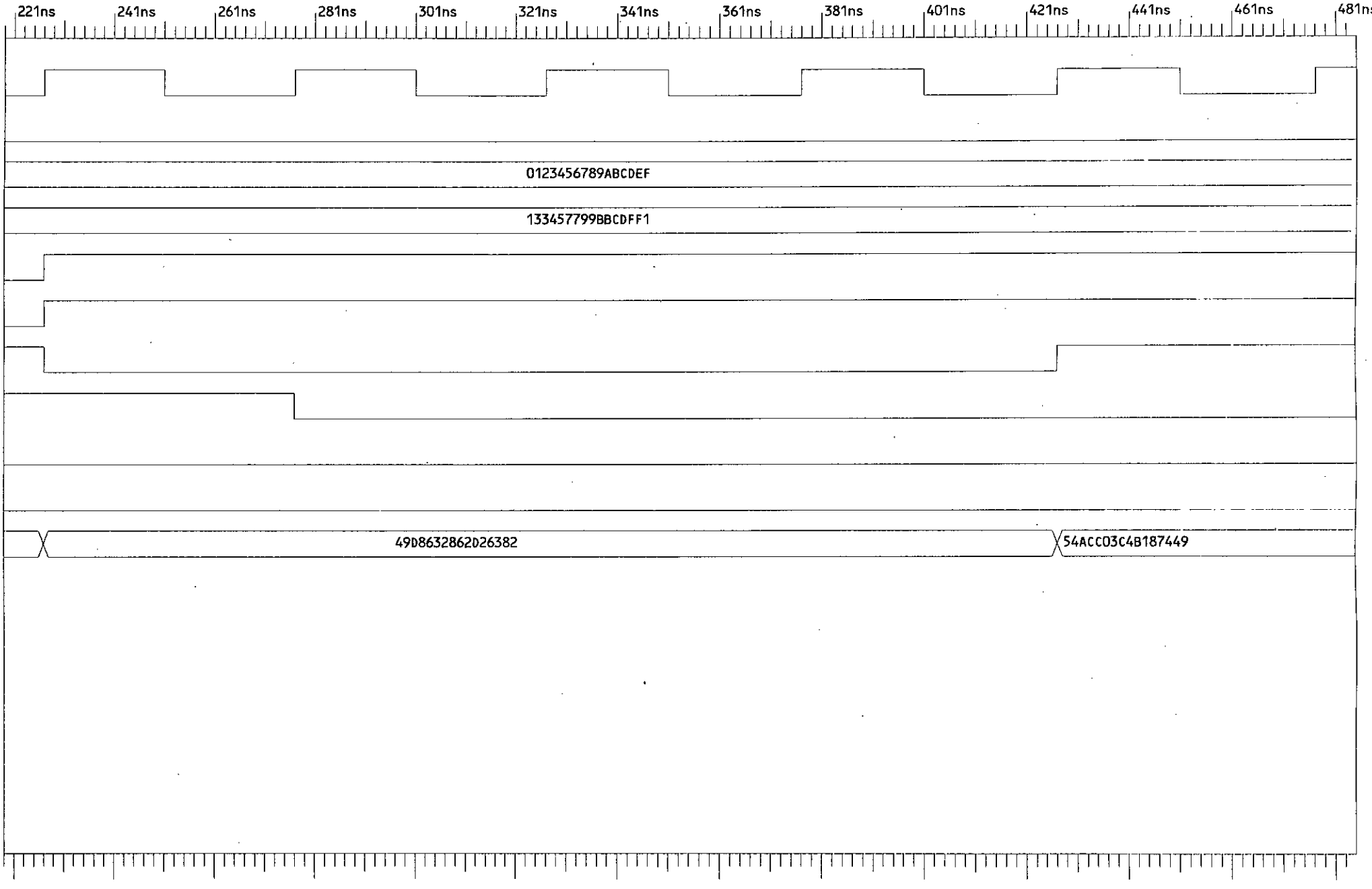



Figure B.5 : Conception DES_VER3.2 (quatre pipeline)


221ns
241ns
261ns
281ns
301ns
321ns
341ns
361ns
381ns
401ns
421ns
441ns
461ns
481ns
0123456789ABCDEF
133457799BBCDFF1
49D8632862D26382
54ACC03C4B187449

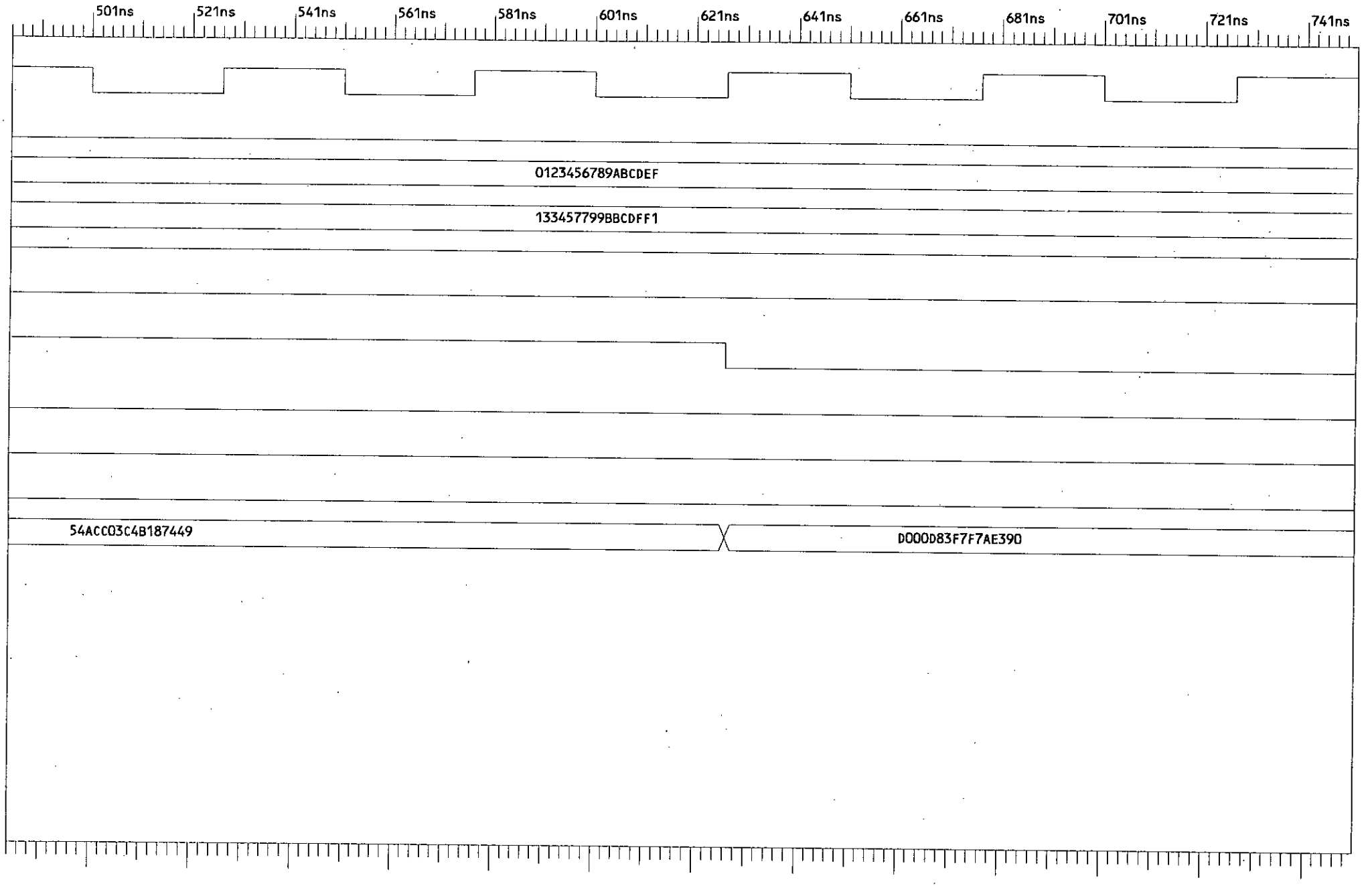



Figure B.5 : Conception DES_VER3.2 (quatre pipeline). suite

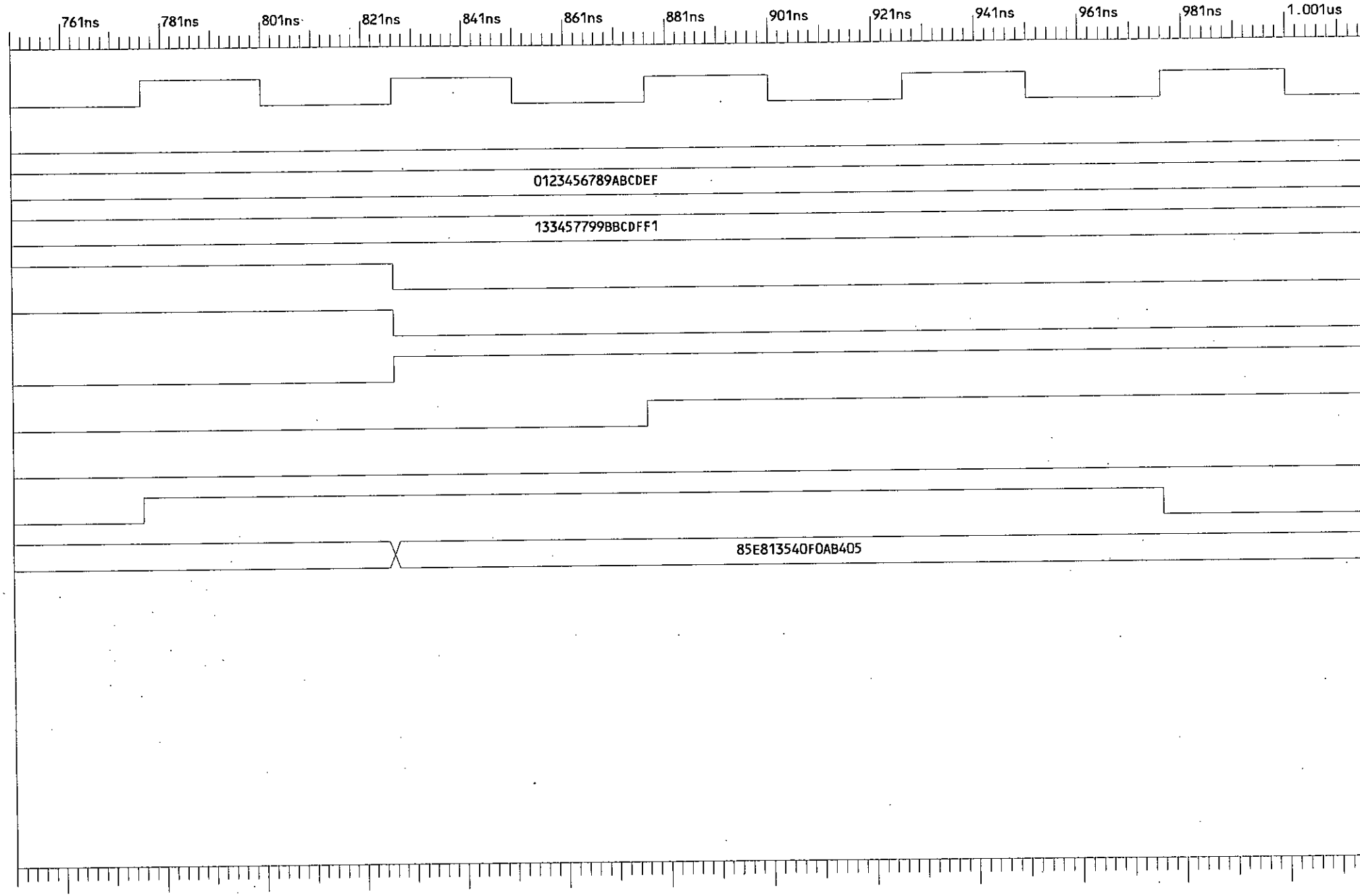



Figure B.5 : Conception DES VER3.2 (quatre pipeline), suite

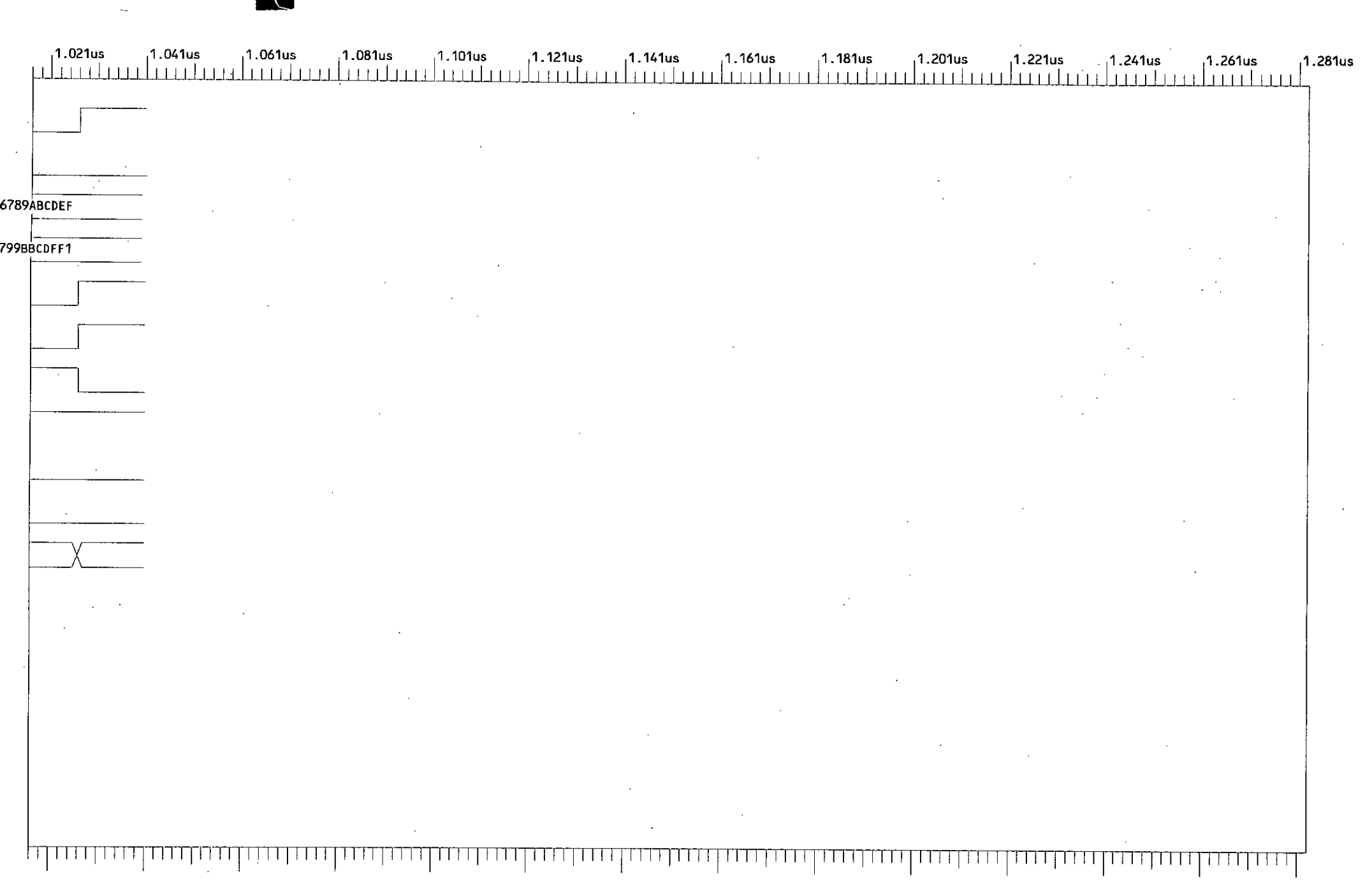



Figure B.5 : Conception DES_VER3.2 (quatre pipeline). suite

# Annexe B.6

La figure B.6 montre le diagramme temporel de l'implémentation DES_VER4 possédant une architecture combinée. La version DES_VER4 est composée d'un pipeline à deux (02) étages et chaque étage du pipeline est formé de deux unités logiques combinatoires (ULCs) concaténées.

La période d'horloge de la simulation est fixée à 50ns. Une fois que le pipeline est rempli avec deux blocs de données il fournit le résultat du chiffrement après huit cycles d'horloge.

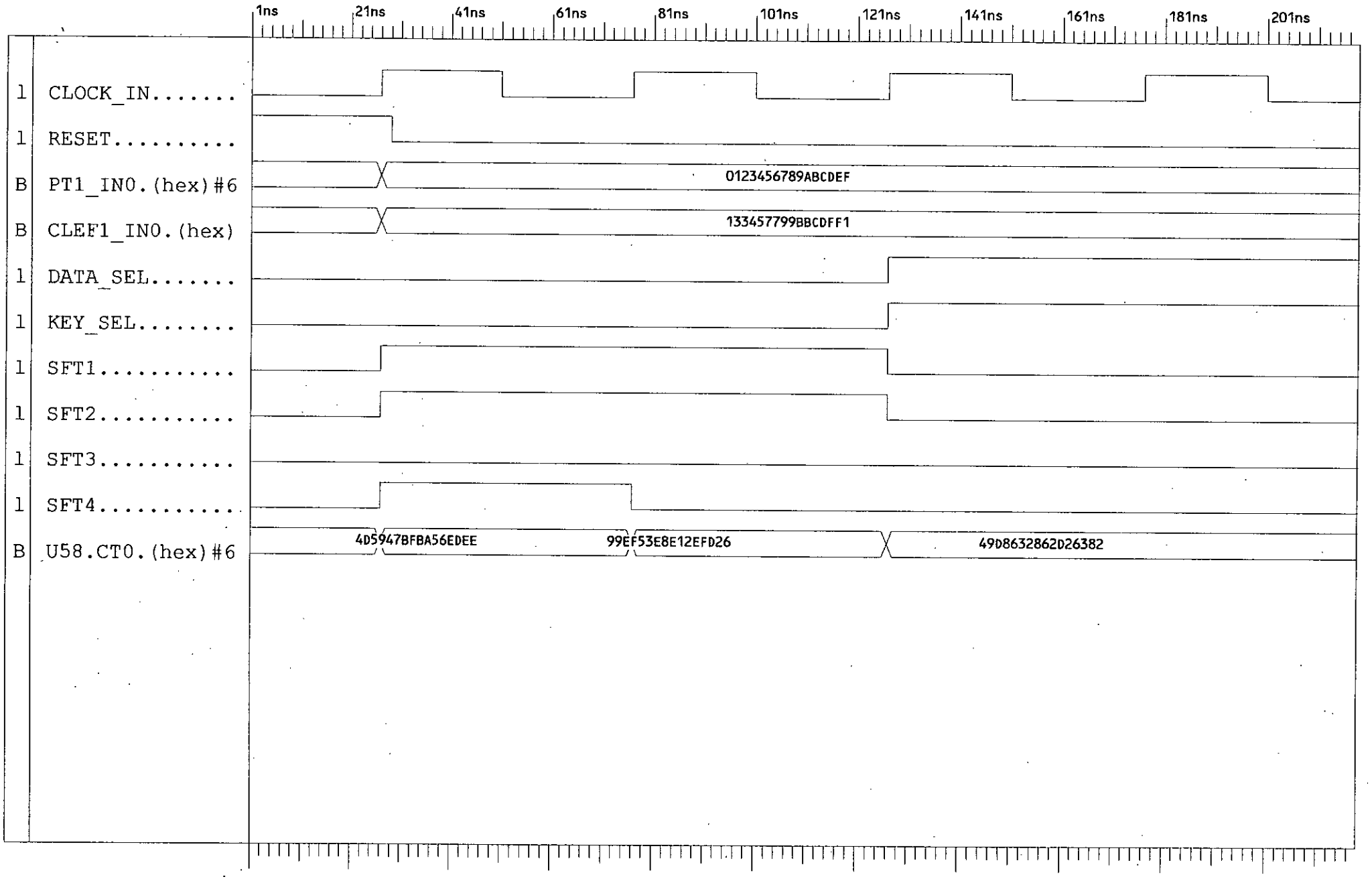



Figure B.6 : Conception DES_VER4 (structure combinée)

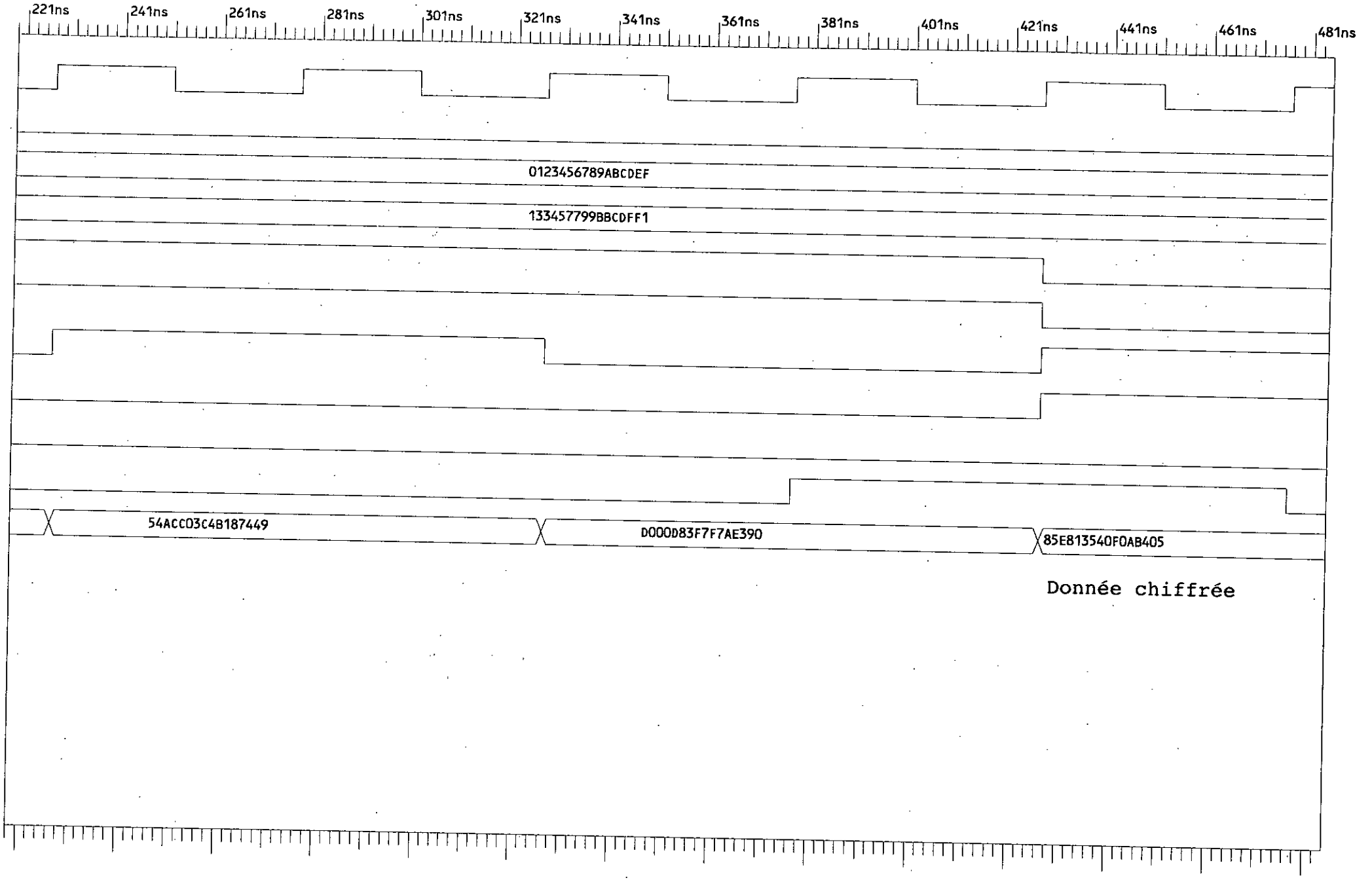



Figure B.6 : Conception DES_VER4 (structure combinée). suite

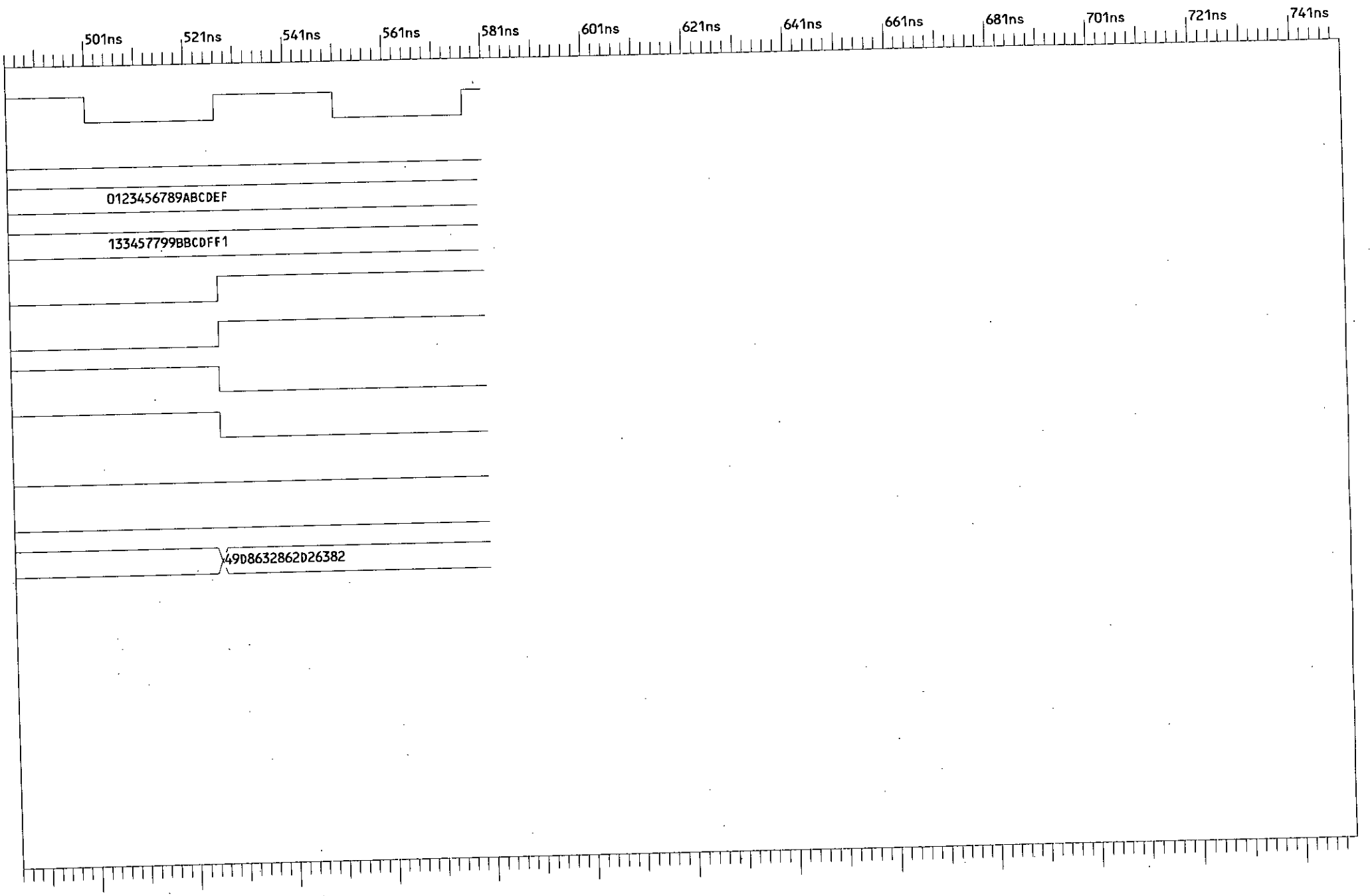



Figure B.6 : Conception DES_VER4 (structure combinée), suite

# Annexe C

## Chronogramme de fonctionnement du cryptoprocesseur

Le chronogramme de la figure C.1 illustre les deux modes de fonctionnement du cryptoprocesseur à savoir, le fonctionnement en mode normal et le fonctionnement en mode interruptible. Ce chronogramme montre aussi les signaux de lecture et d'écriture (WE_ et OE_) du cryptoprocesseur.

Les différents signaux constituants le chronogramme de fonctionnement du cryptoprocesseur sont obtenus grâce à l'exécution du programme suivant :

| | Adresse | Encodage | Mnémonique | |
|---|---|---|---|---|
| | 0000 | D0 04 | jump INT | ; saut vers l'adresse du programme prioritaire (vecteur d'interruption) |
| | 0002 | 40 FC | load #FC | ; chargement de l'accumulateur avec la valeur FC |
| INT | 0004 | E8 | reti | ; retour au programme interrompu |

- *Fonctionnement en mode normal* : durant l'étape 1 de la figure C.1, le cryptoprocesseur fonctionne en mode normal. Il commence à lire la première instruction depuis l'adresse 2 (voir signal PC et sortie de la mémoire programme (OUT_RAM_PROG) du chronogramme) du segment programme de la mémoire. Au cours de cette étape il y a apparition d'une interruption sur la broche INT (signal INT). Cette interruption est enregistrée par le circuit de traitement d'interruption en maintenant le signal INTRPT à l'état haut. A la fin du traitement de l'instruction en cours, le cryptoprocesseur franchi l'étape 2 durant laquelle il vérifie l'état des signaux INTRPT et IFLAG qui sont dans ce cas respectivement à 1 et à 0. Par conséquent, le cryptoprocesseur rentre en mode interruptible.

- *Fonctionnement en mode interruptible* : le cryptoprocesseur commence par mettre le signal INTRPT à 0 et le signal IFLAG à 1 (voir figure C.1), il sauvegarde ensuite le contenu de l'accumulateur et du compteur programme dans la pile (l'entrée de la pile est représentée par le signal IN_PILE) et en incrémentant à chaque fois le pointeur de pile (signal SP).
  A l'adresse 0 du segment programme, l'instruction jump indique au cryptoprocesseur l'adresse du début du programme prioritaire dans notre exemple c'est l'adresse 4. Le programme d'exemple contient uniquement l'instruction de retour au programme interrompu (reti). Cette instruction restaure le contenu de la pile dans l'accumulateur et dans le compteur programme (voir signal de sortie de pile (OUT_PILE)) tout en décrémentant la valeur du pointeur de pile.

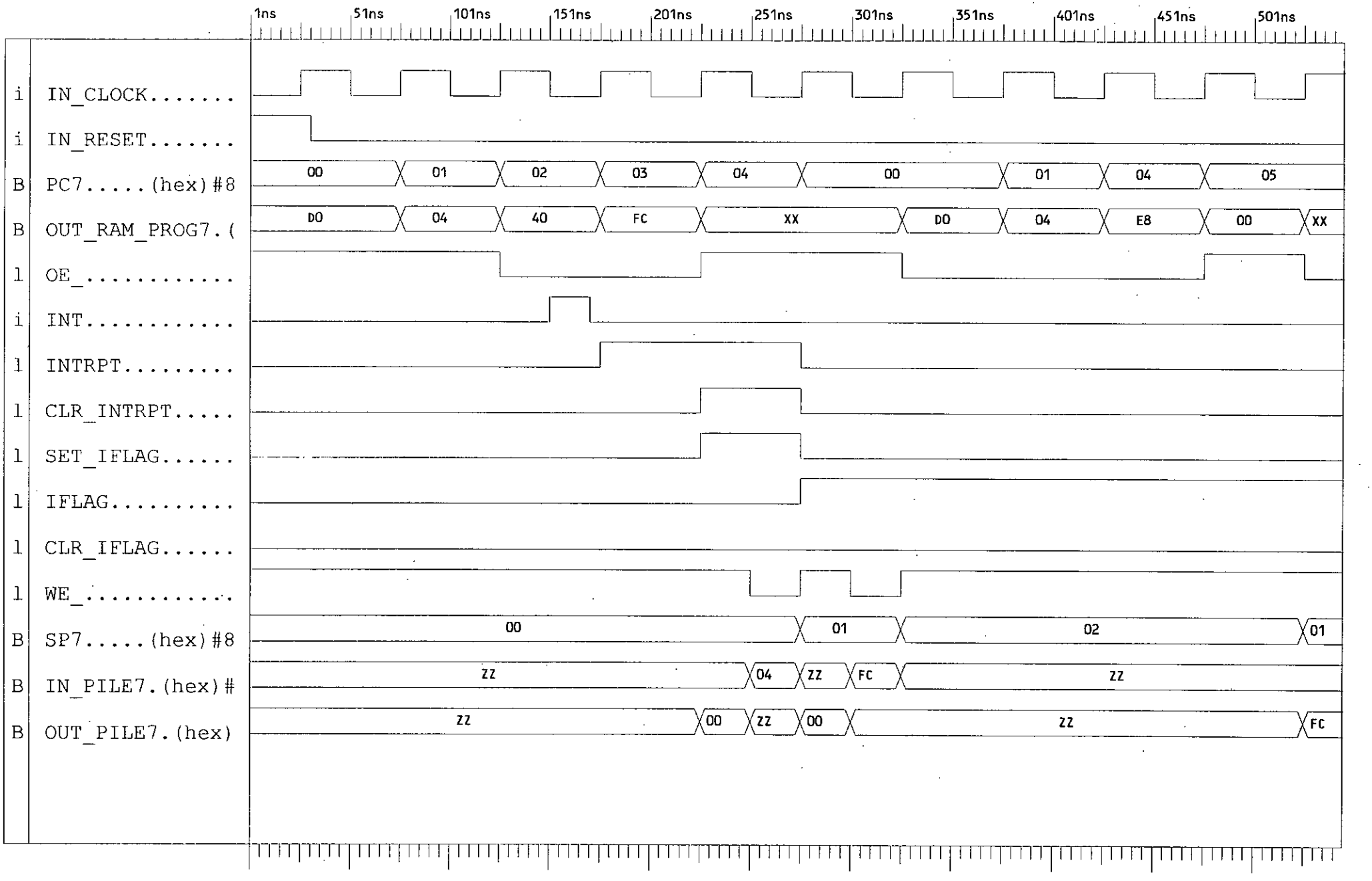



Figure C.1 : Fonctionnement du cryptoprocesseur en mode d'adressage interruptible

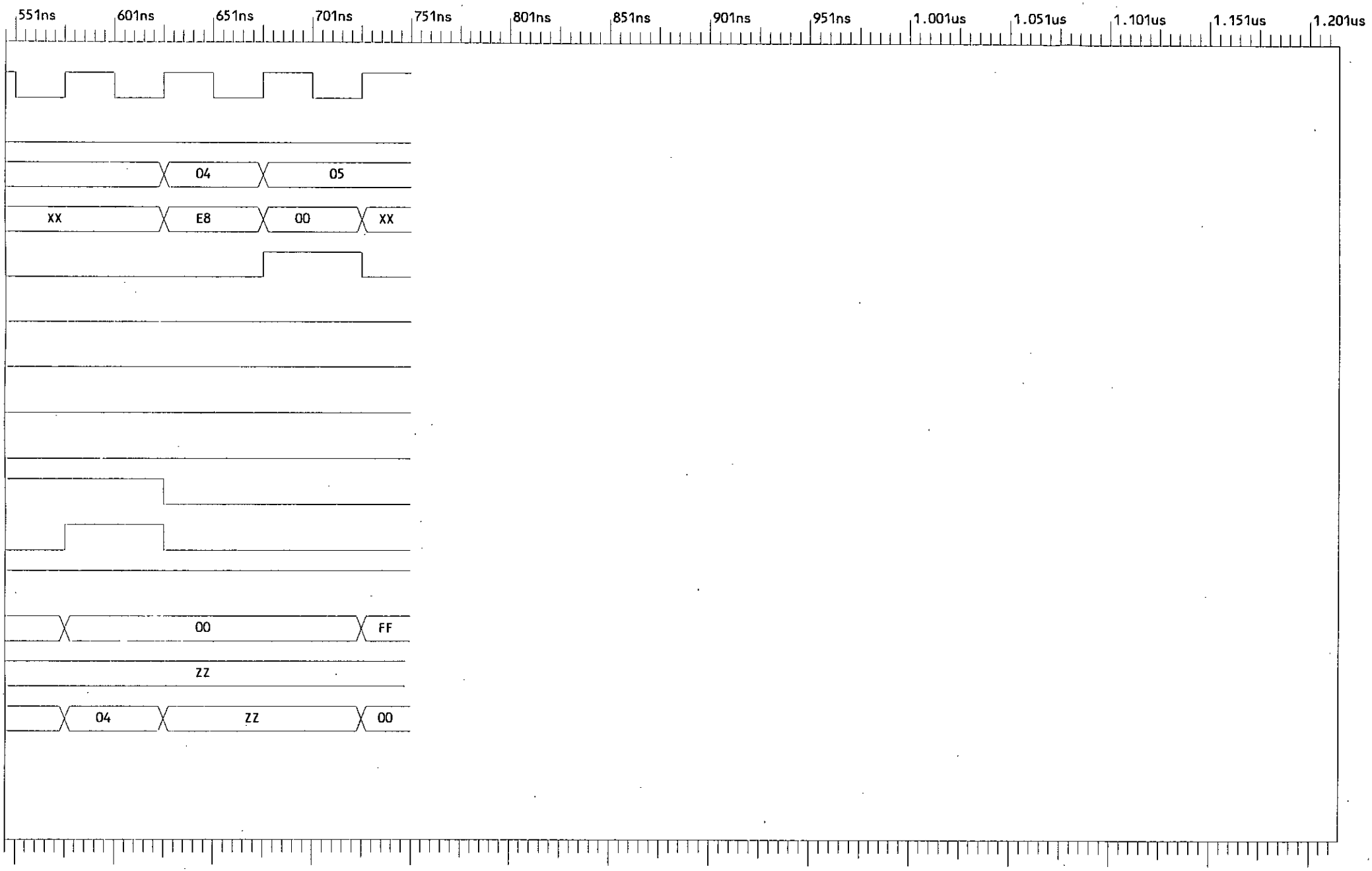



Figure C.1 : Fonctionnement du cryptoprocesseur en mode d'adressage interruptible (suite)

# Annexe D

Fonctionnement du cryptoprocesseur selon le mode d'adressage.

## a) Adressage indexé

L'instruction load (Rd) utilise un mode d'adressage indexé. Cette instruction est illustrée par le programme donné en ci-dessous. Le déroulement du programme est donné par la figure D.1.

```
0002    40 02    load #2     ;} charger l'adresse de la case mémoire du
0004    01       store R1    ;} segment DATA dans le registre R1.
0005    29       load (R1)   ; charger l'accumulateur avec le contenu de la
                             case mémoire du segment DATA, dont
                             l'adresse se trouve dans le registre R1.
```

## b) Adressage immédiat

L'instruction load #d illustre le mode d'adressage immédiat. L'exécution de cette instruction par le cryptoprocesseur est illustré par la figure D.2.

```
0002    40 02    load #2     ; charger l'accumulateur avec la valeur 02.
                             Cette valeur se trouve dans la partie
                             opérande de l'instruction.
```

## c) Adressage direct

Ce mode d'adressage est illustré par l'exécution du cryptoprocesseur de l'instruction load d. La figure D.3 montre le déroulement du programme suivant.

```
0002    30 02    load 2      ; charger l'accumulateur avec le contenu de la
                             case mémoire du segment DATA d'adresse2.
```

## d) Adressage indirect

L'instruction load (d) illustre le mode d'adressage indirect. Dans cet exemple la case mémoire du segment DATA, dont le contenu sera chargé dans l'accumulateur, possède l'adresse 02. La donnée stockée dans cette adresse est la valeur 45. Le déroulement du programme suivant est donné par la figure D.4

```
0002    38 03    load 3      ; charger l'accumulateur avec le contenu de la
                             case mémoire du segment DATA dont
                             l'adresse se trouve dans la case mémoire du
                             segment DATA d'adresse d (d=3).
```

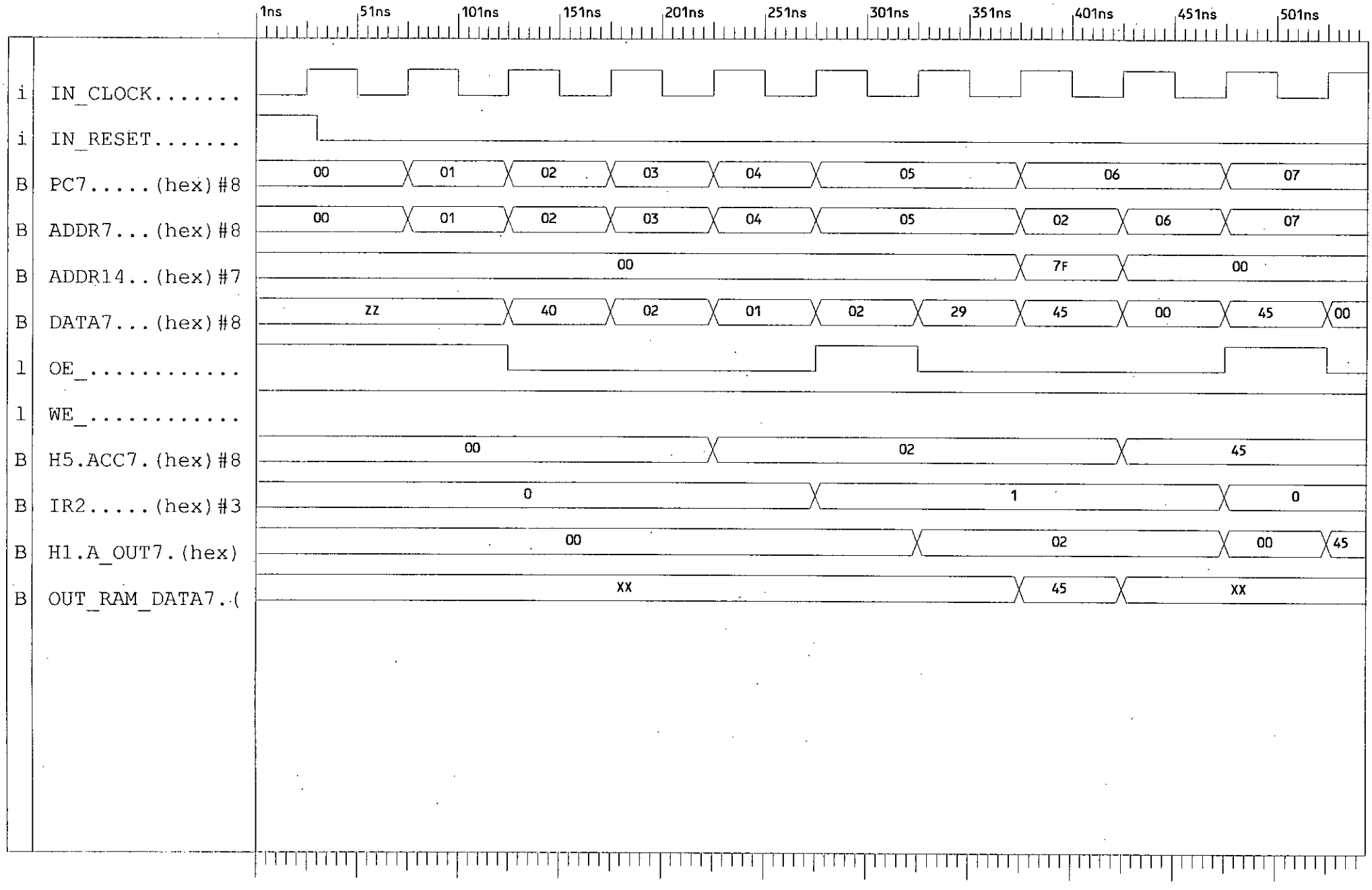



Figure D.1: Fonctionnement du cryptoprocesseur en mode d'adressage indexé

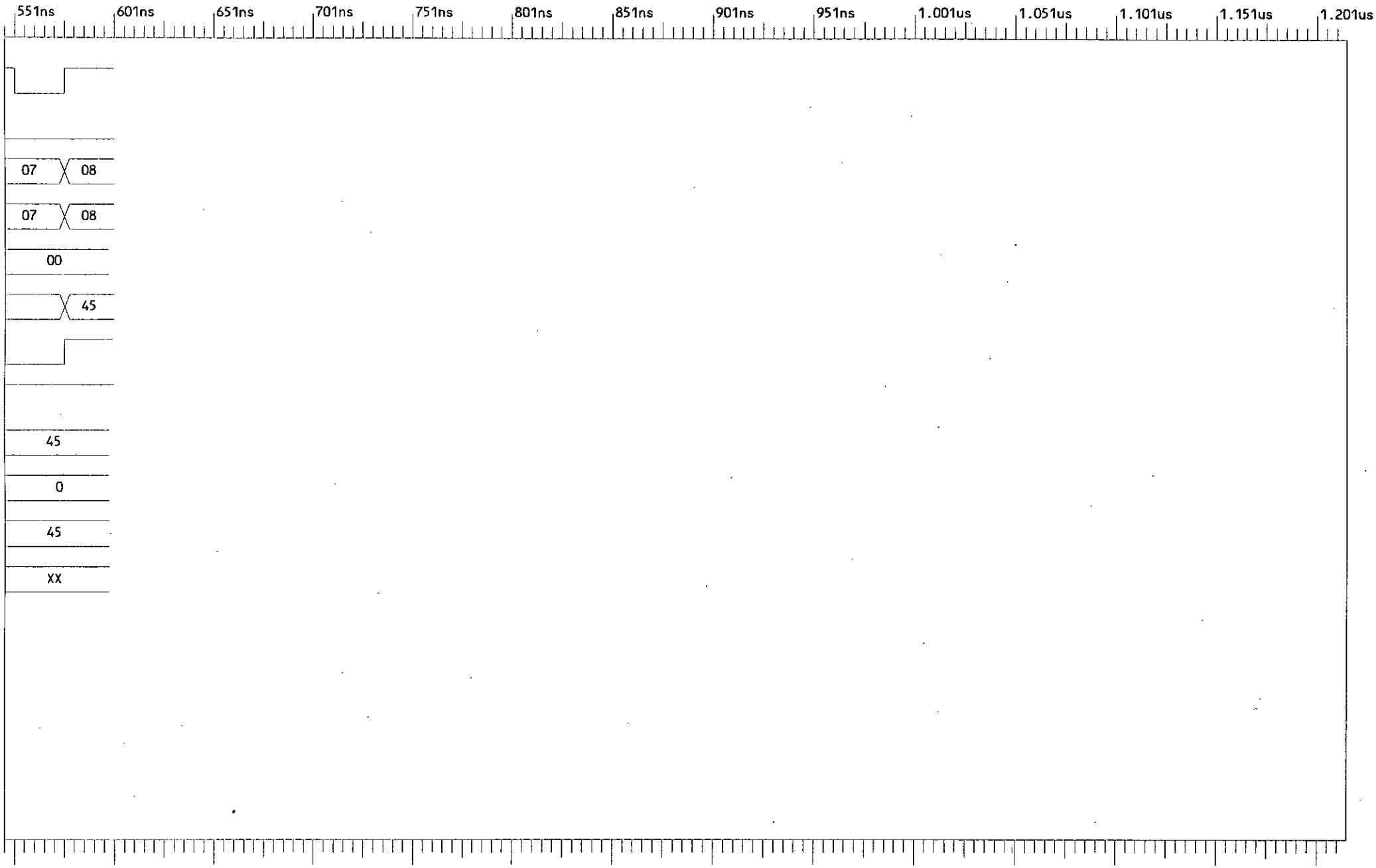



Figure D.1 : Fonctionnement du cryptoprocesseur en mode d'adressage indexé (suite)

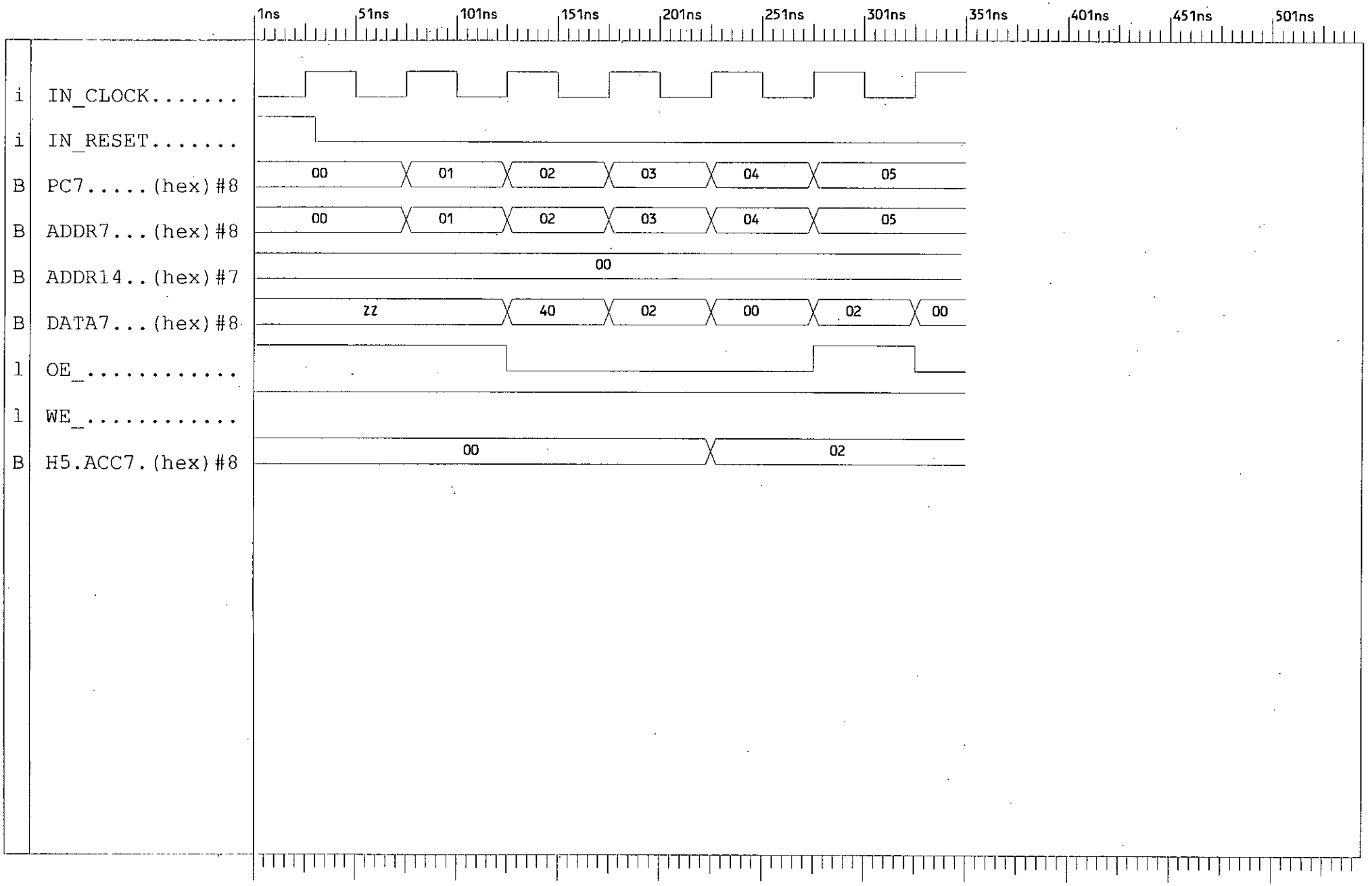



Figure D.2 : Fonctionnement du cryptoprocesseur en mode d'adressage immédiat

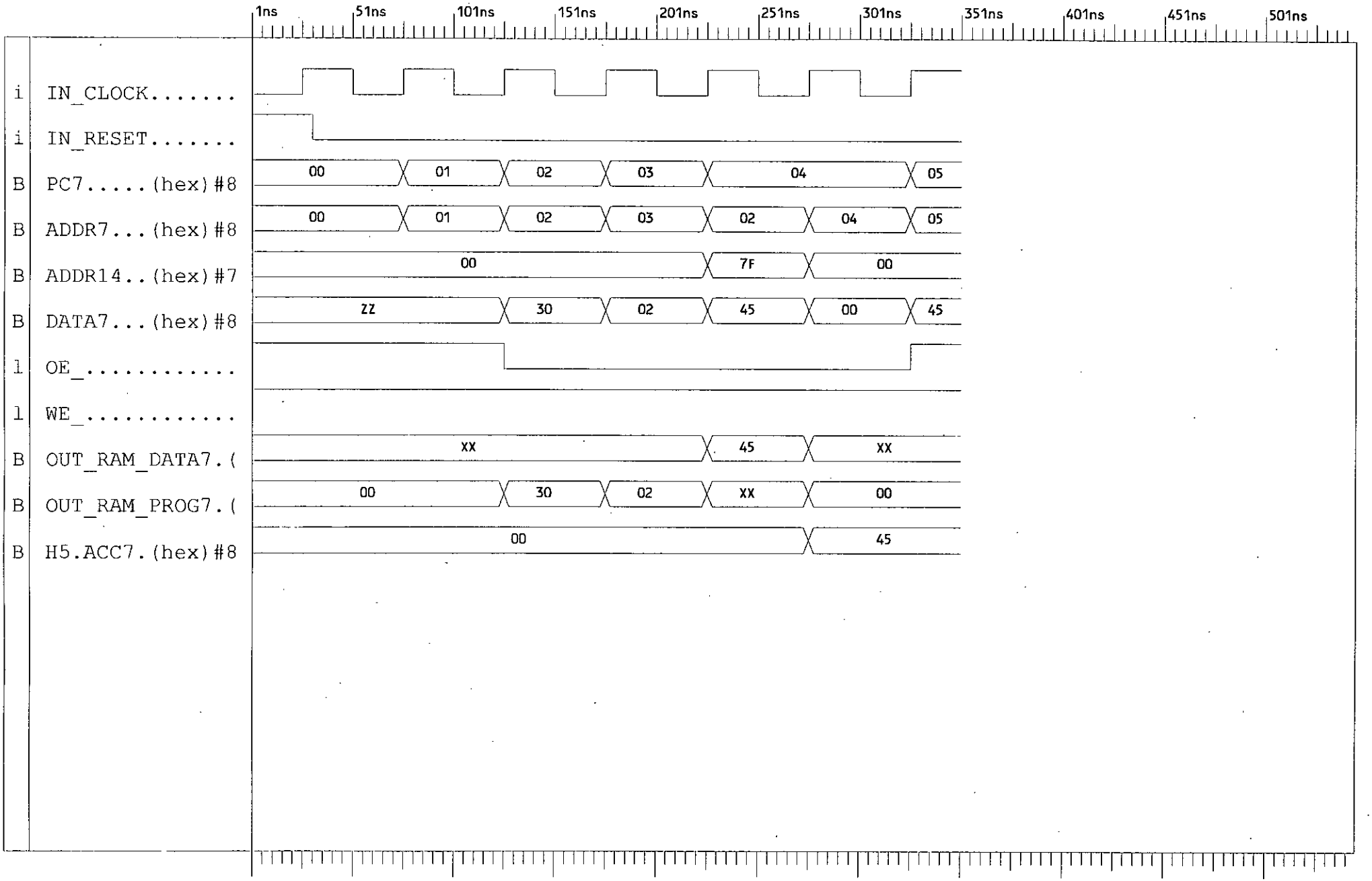



Figure D.3 : Fonctionnement du cryptoprocesseur en mode d'adressage direct

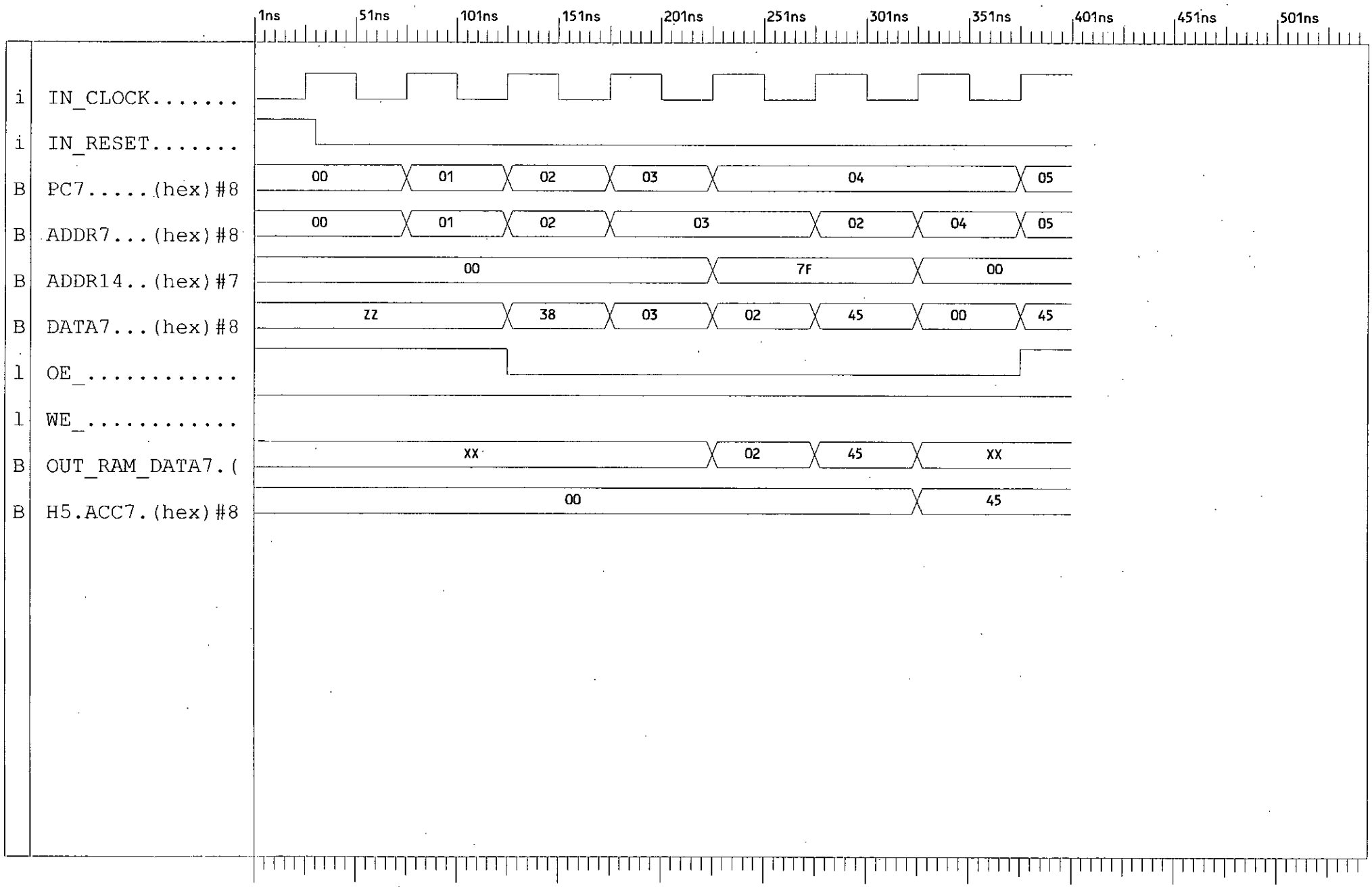


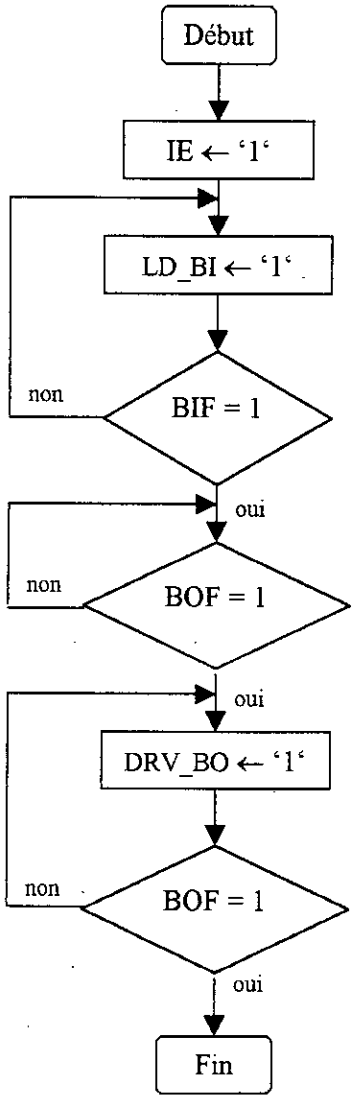

Figure D.4 : Fonctionnement du cryptoprocesseur en mode d'adressage indirect

# Annexe E

Le diagramme temporel de la figure E.1 illustre la procédure de chiffrement d'un bloc de données de 64 bits stocké dans le segment DATA de la mémoire. Le résultat du chiffrement sera envoyé vers le port de sortie du cryptoprocesseur. La clef correspondante est stockée dans une mémoire morte "ROM" située à l'intérieur du cryptoprocesseur.

Dans cet exemple nous choisissons comme texte clair le bloc (hexadécimal) 0123456789ABCDEF et la clef correspondante le bloc (hexadécimal) 133457799BBCDFF1

Le programme se déroule selon l'organigramme suivant :




; début de programme

;Validation du tampon d'entrée

;Chargement des données dans le tampon d'entrée par bloc de 8 bits

; test si le tampon d'entrée est plein

; test si le tampon de sortie est plein

; écriture du contenu du tampon de sortie sur le port de sortie du cryptoprocesseur

; test si tampon de sortie est vide

; fin de programme

Programme correspondant

| | Adresse | Mnémonique | Instruction |
|---|---|---|---|
| | 0002 | 40 00 | load #0 |
| | 0004 | 01 | store R1 |
| | 0005 | 50 | initbuf |
| chiffrer | 0006 | F1 | encm(R1) |
| | 0007 | 40 01 | load #1 |
| | 0009 | A0 | clear_c |
| | 000A | 89 | add R1 |
| | 000B | 01 | store R1 |
| | 000C | 58 10 | jbif |
| | 000E | D0 06 | jump chiffrer |
| test | 0010 | 70 14 | jbof |
| | 0012 | D0 10 | jump test |
| | 0014 | 78 | out_bo |
| | 0015 | 70 14 | jbof |
| halt | 0017 | D0 17 | jump halt |

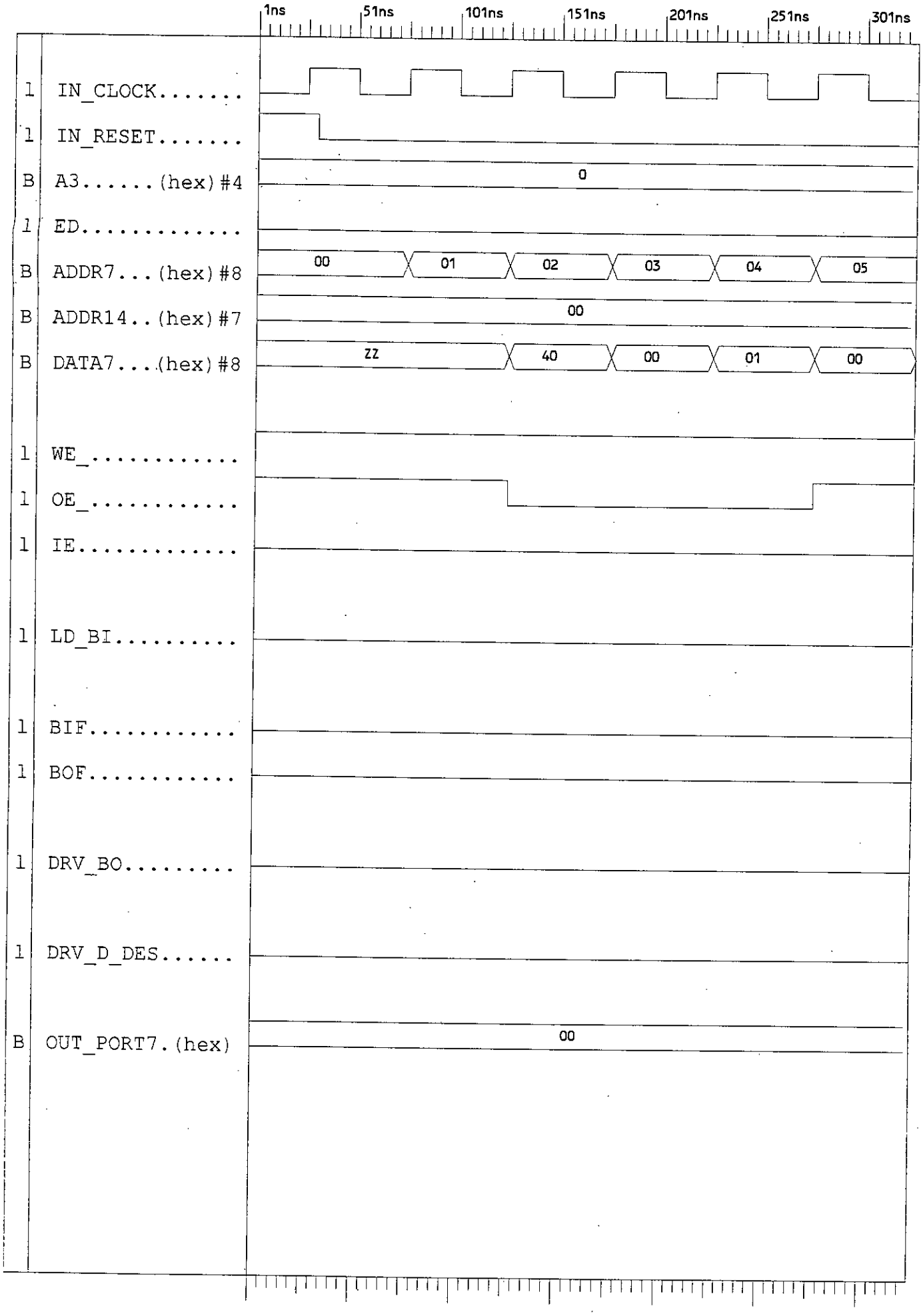



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

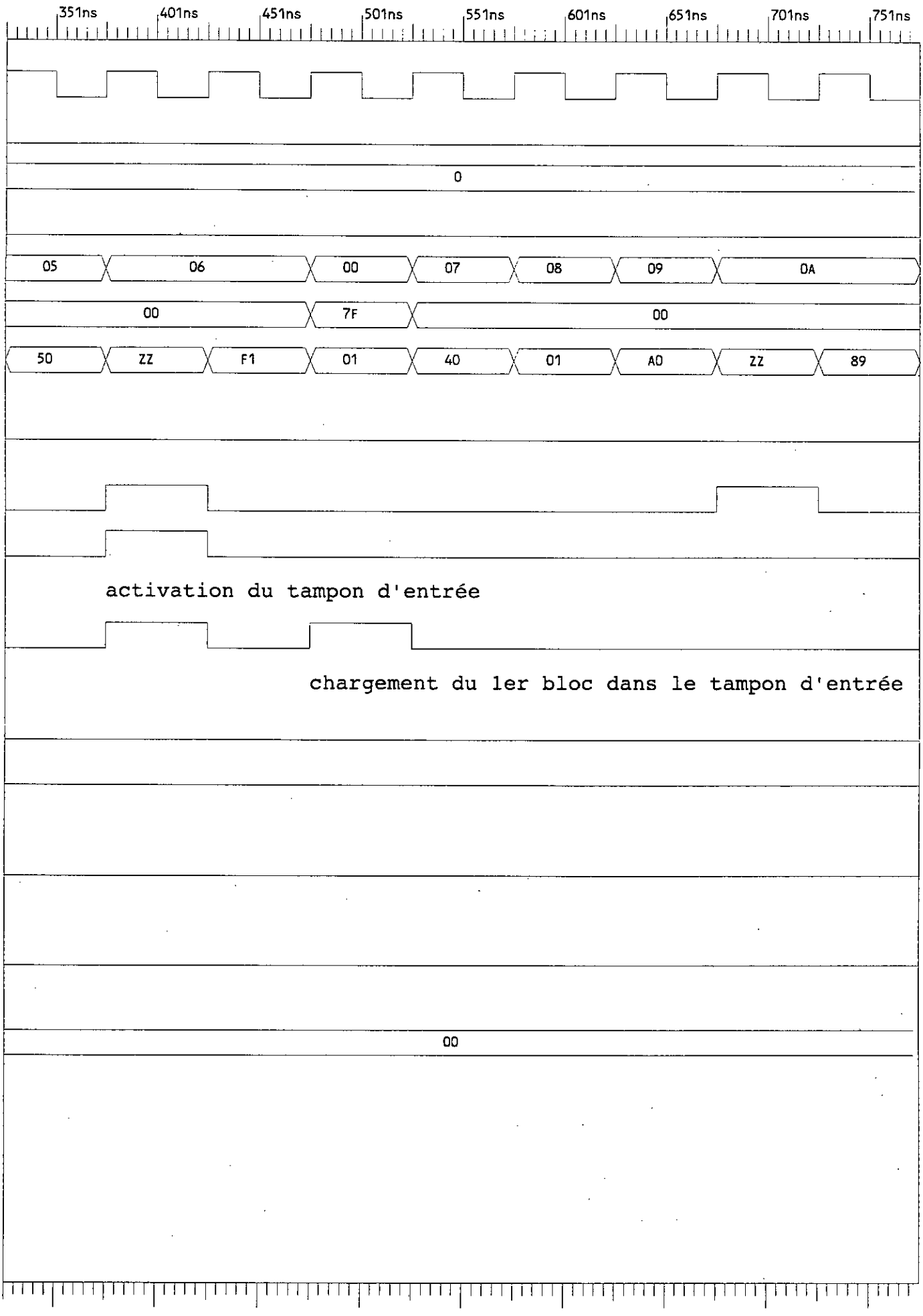



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur (suite).

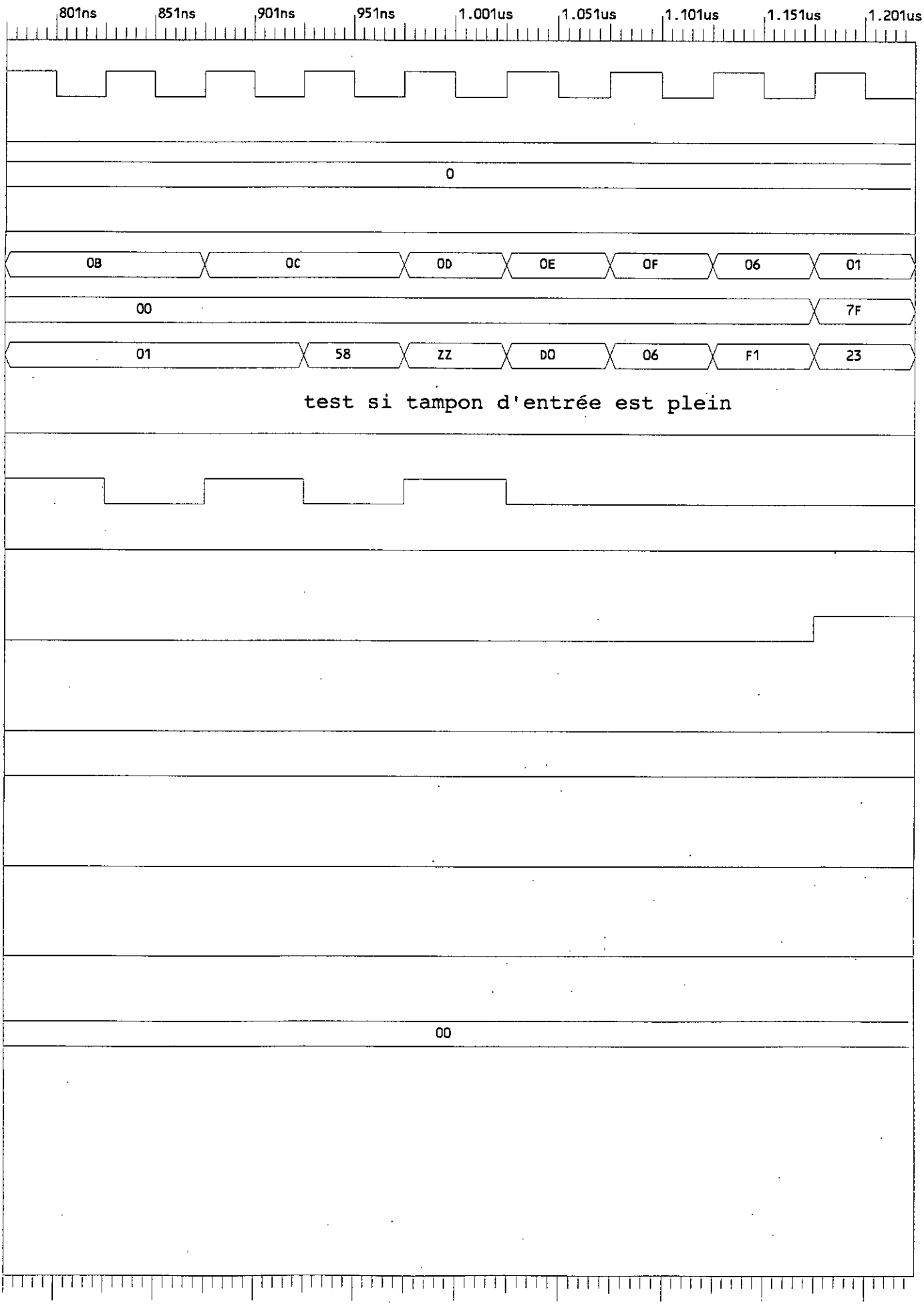



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

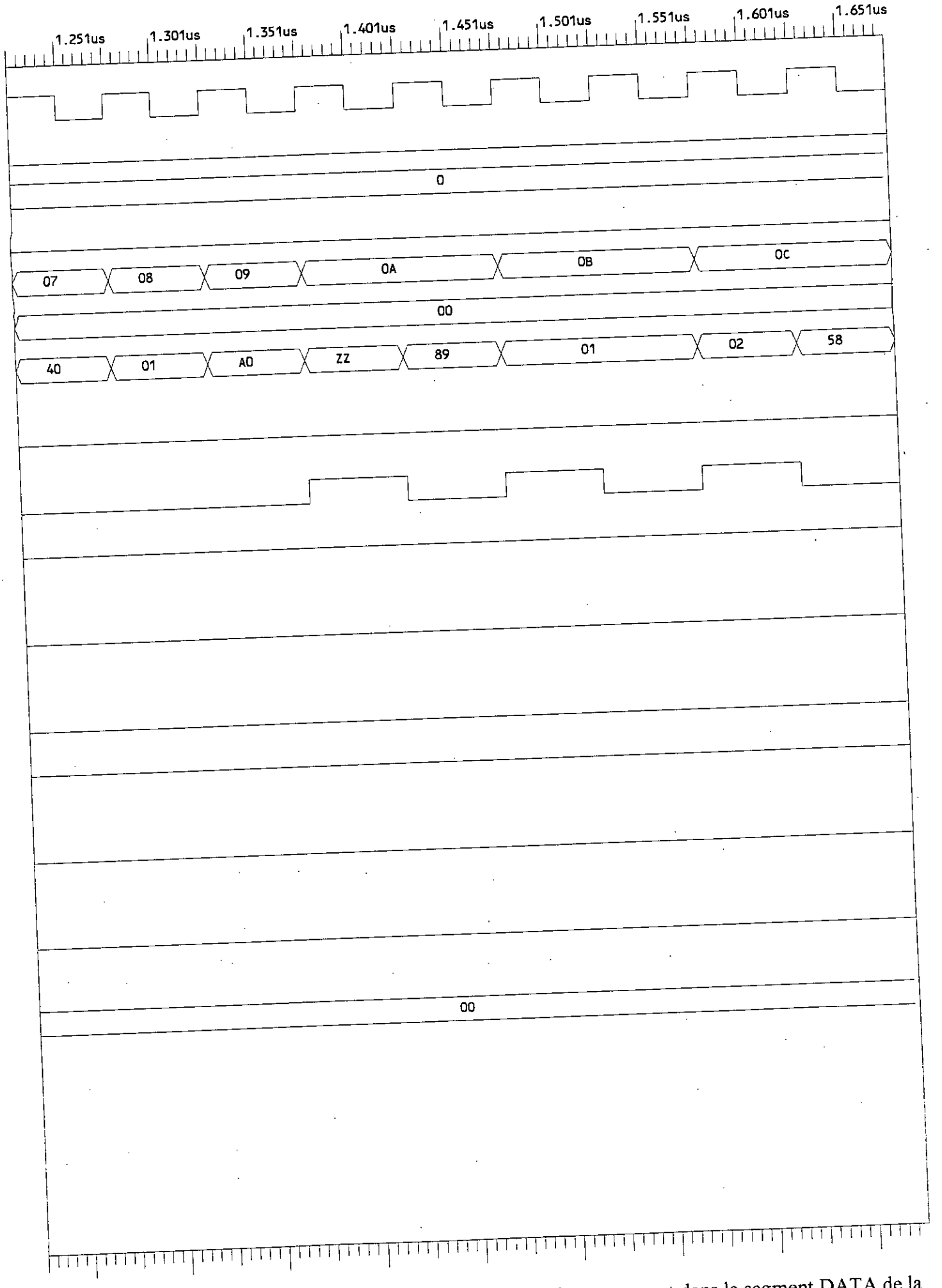



Figure F : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

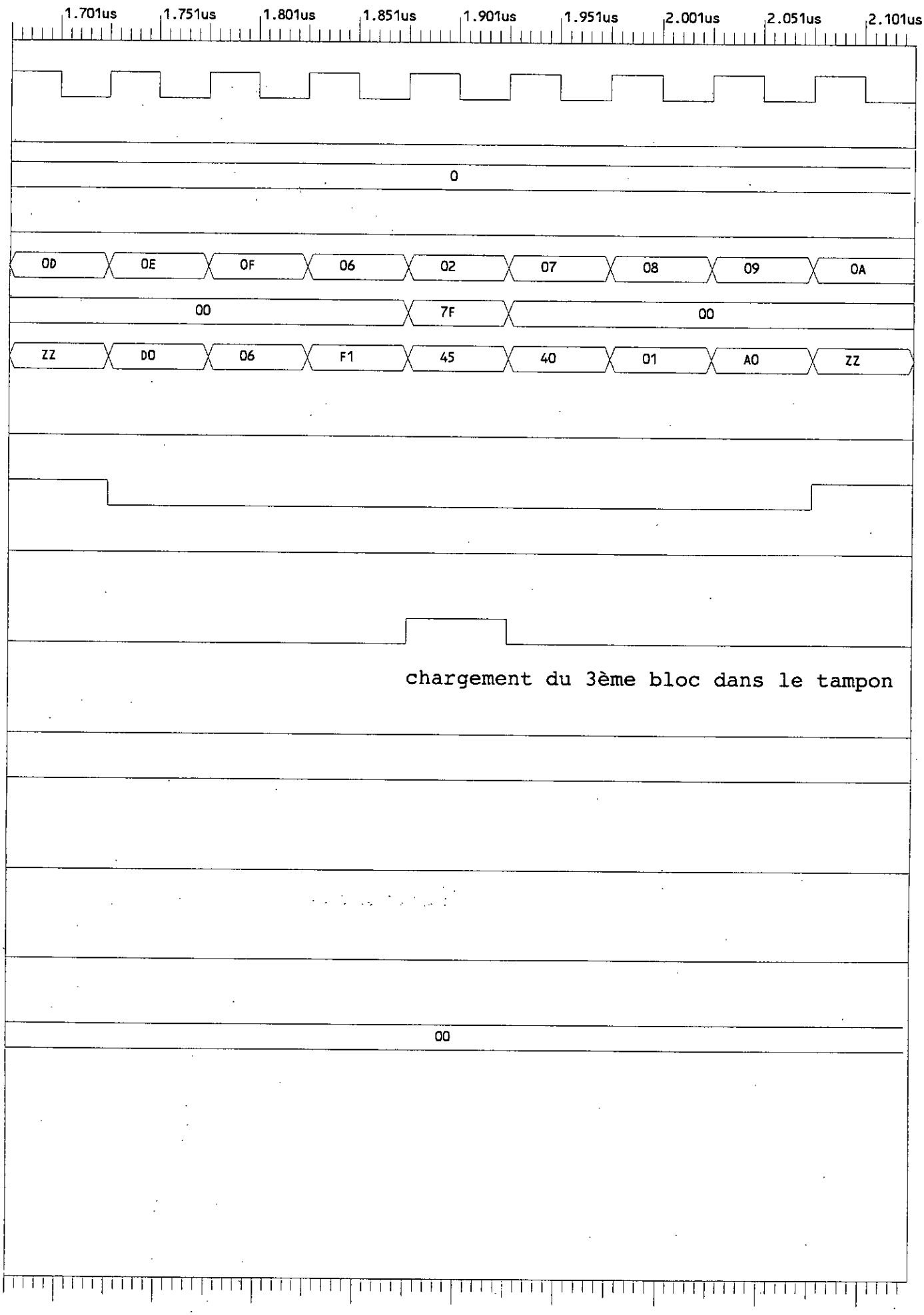



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

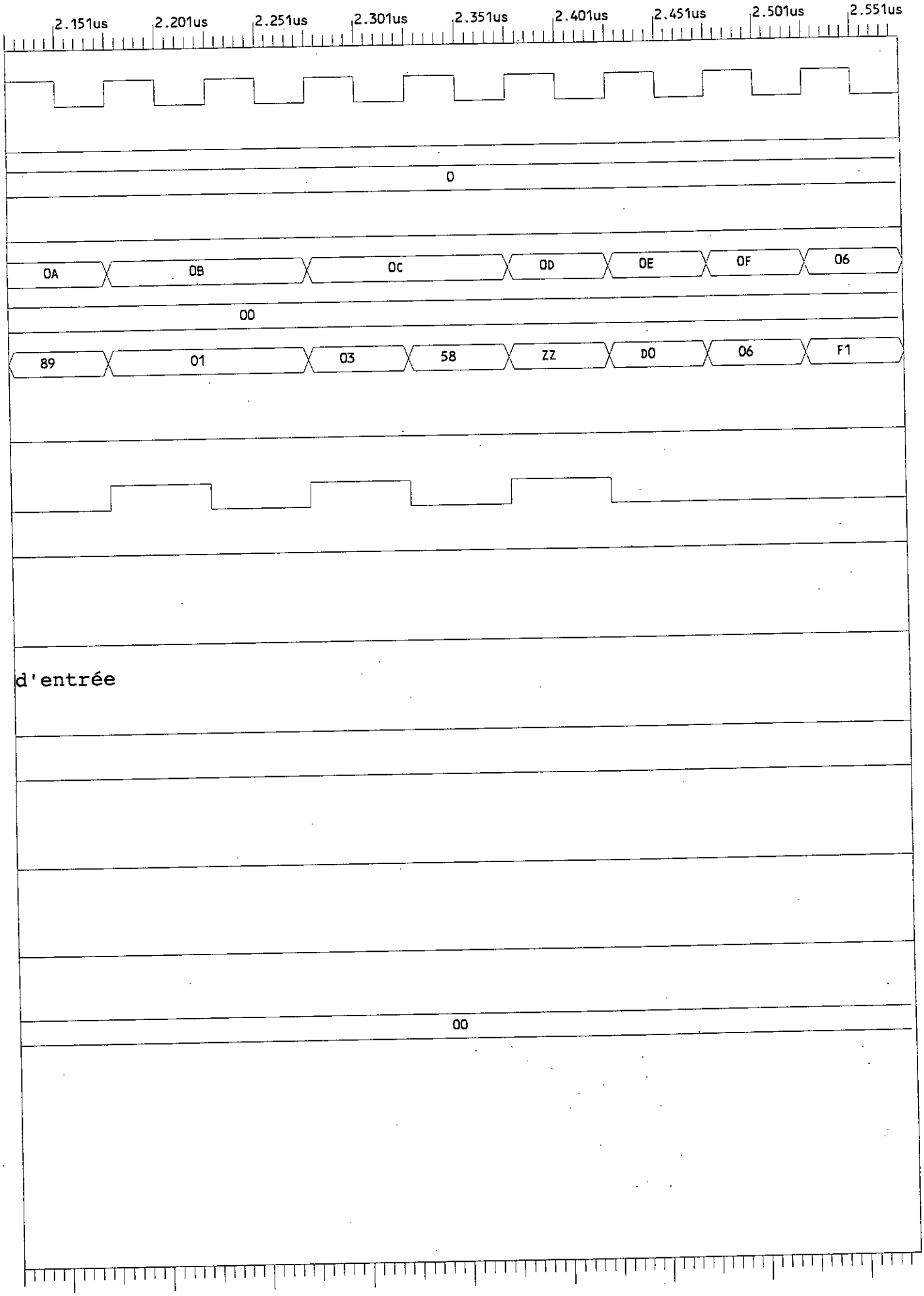
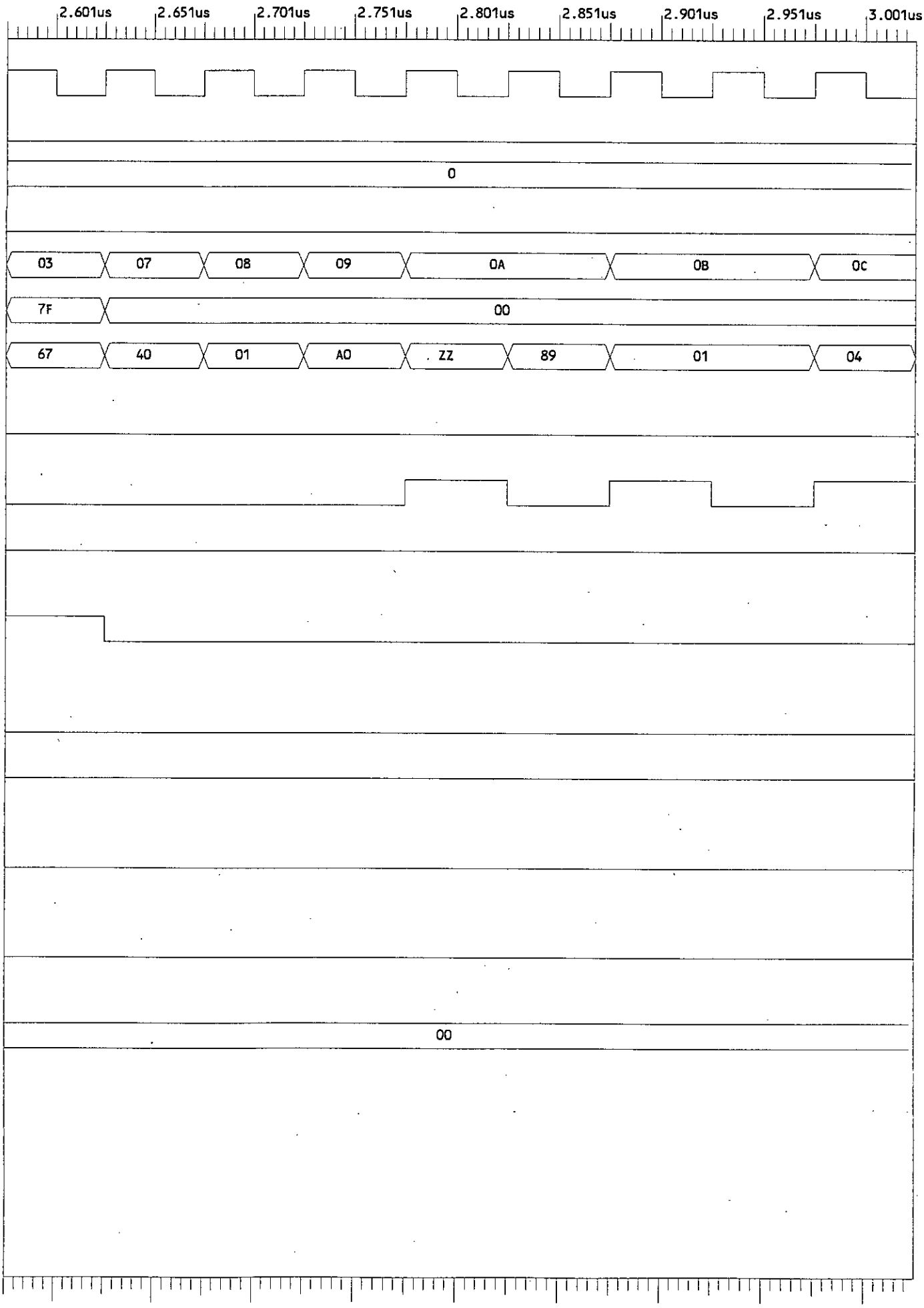



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la


2.601us
2.651us
2.701us
2.751us
2.801us
2.851us
2.901us
2.951us
3.001us
0
03
07
08
09
0A
0B
0C
7F
00
67
40
01
A0
ZZ
89
01
04
00

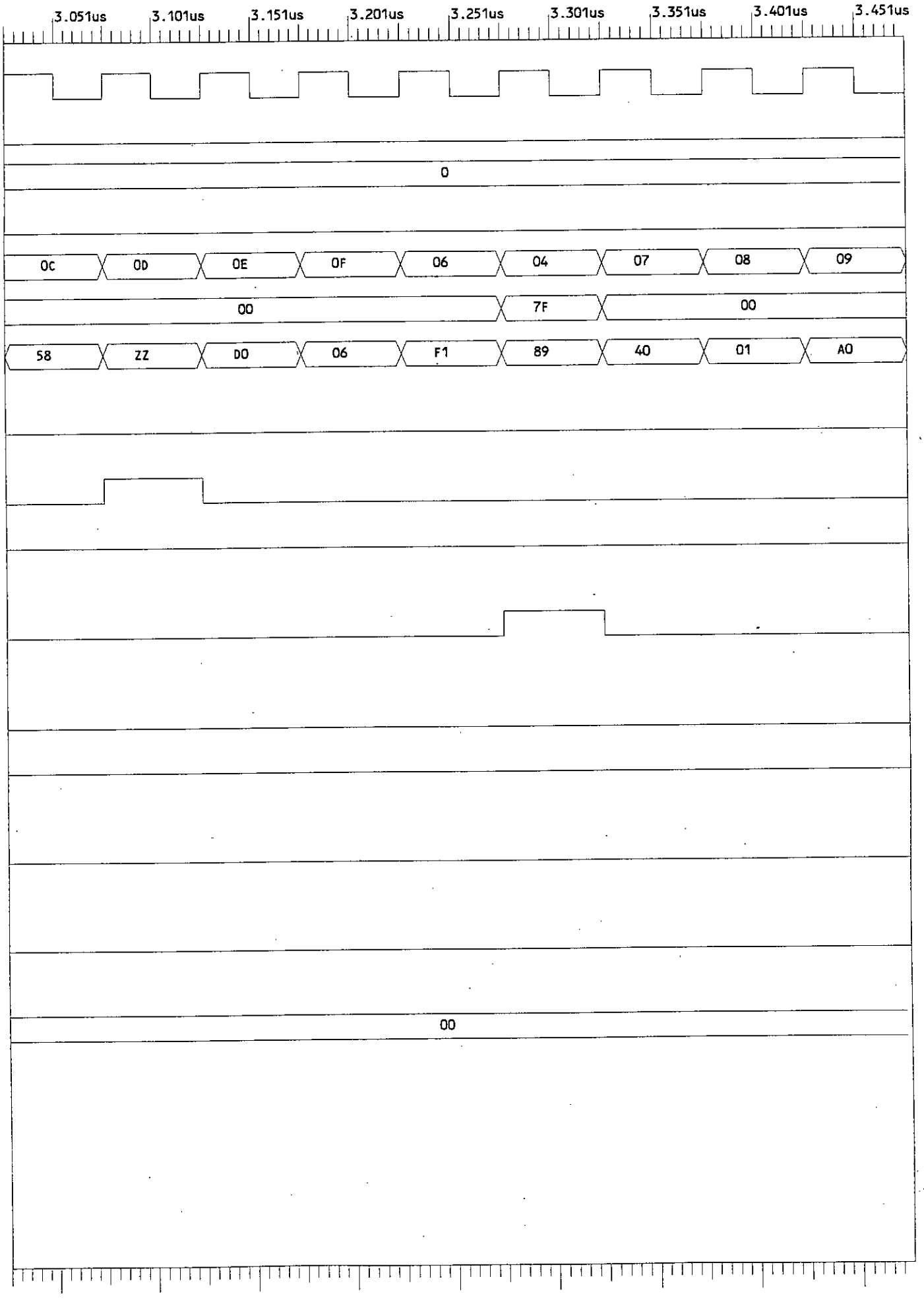



Figure F : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

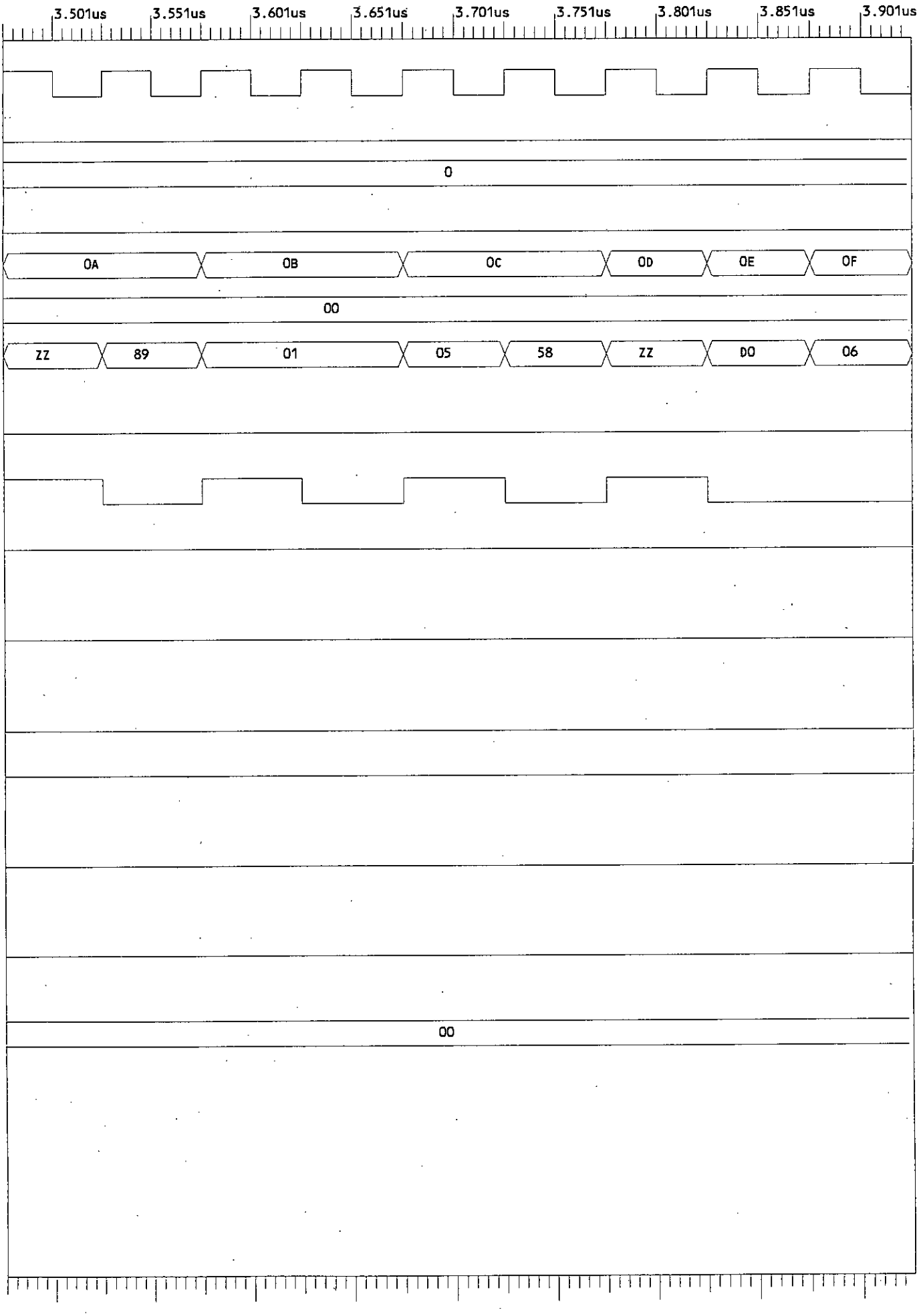



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

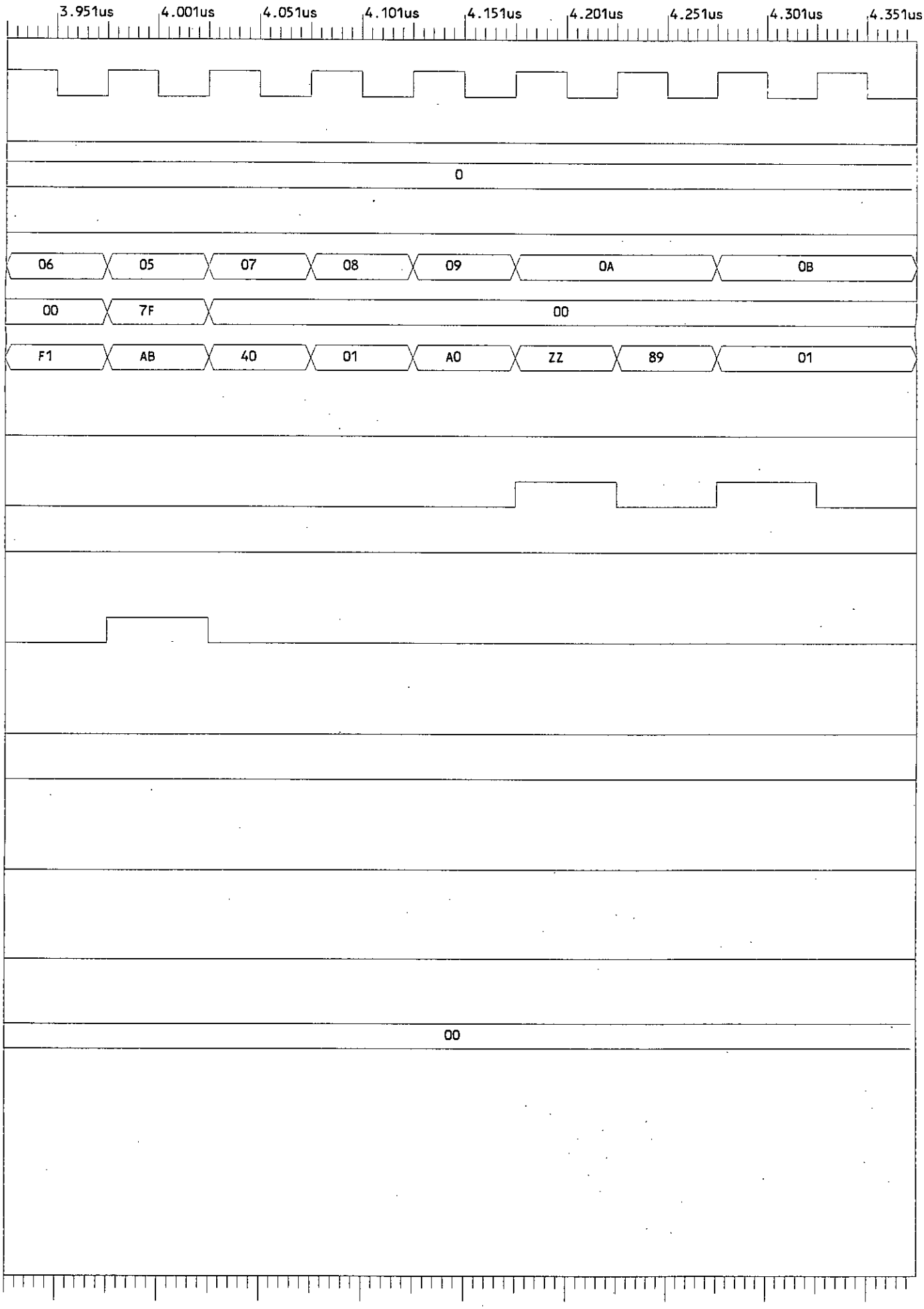



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur (suite).

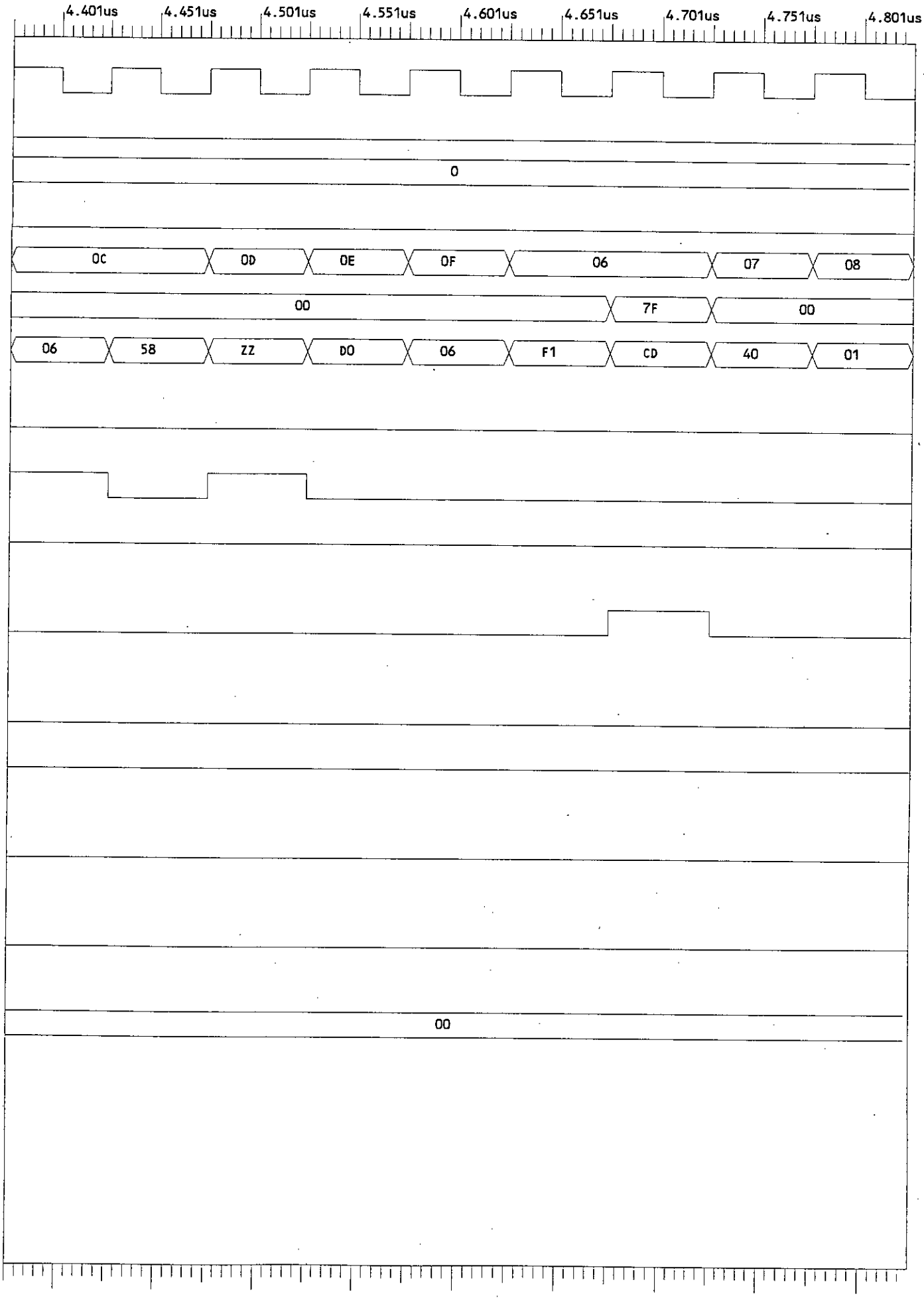



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

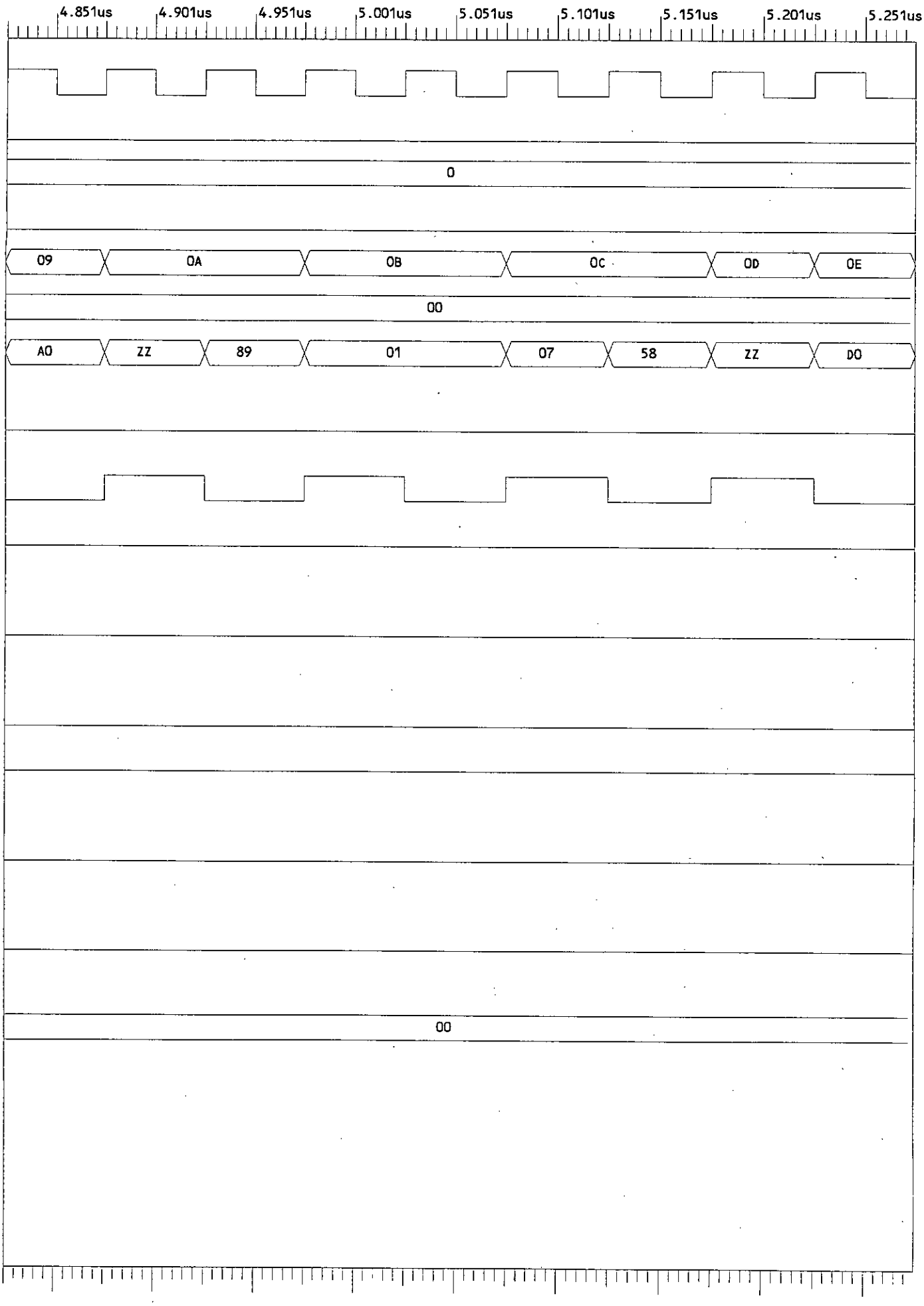



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur (suite)

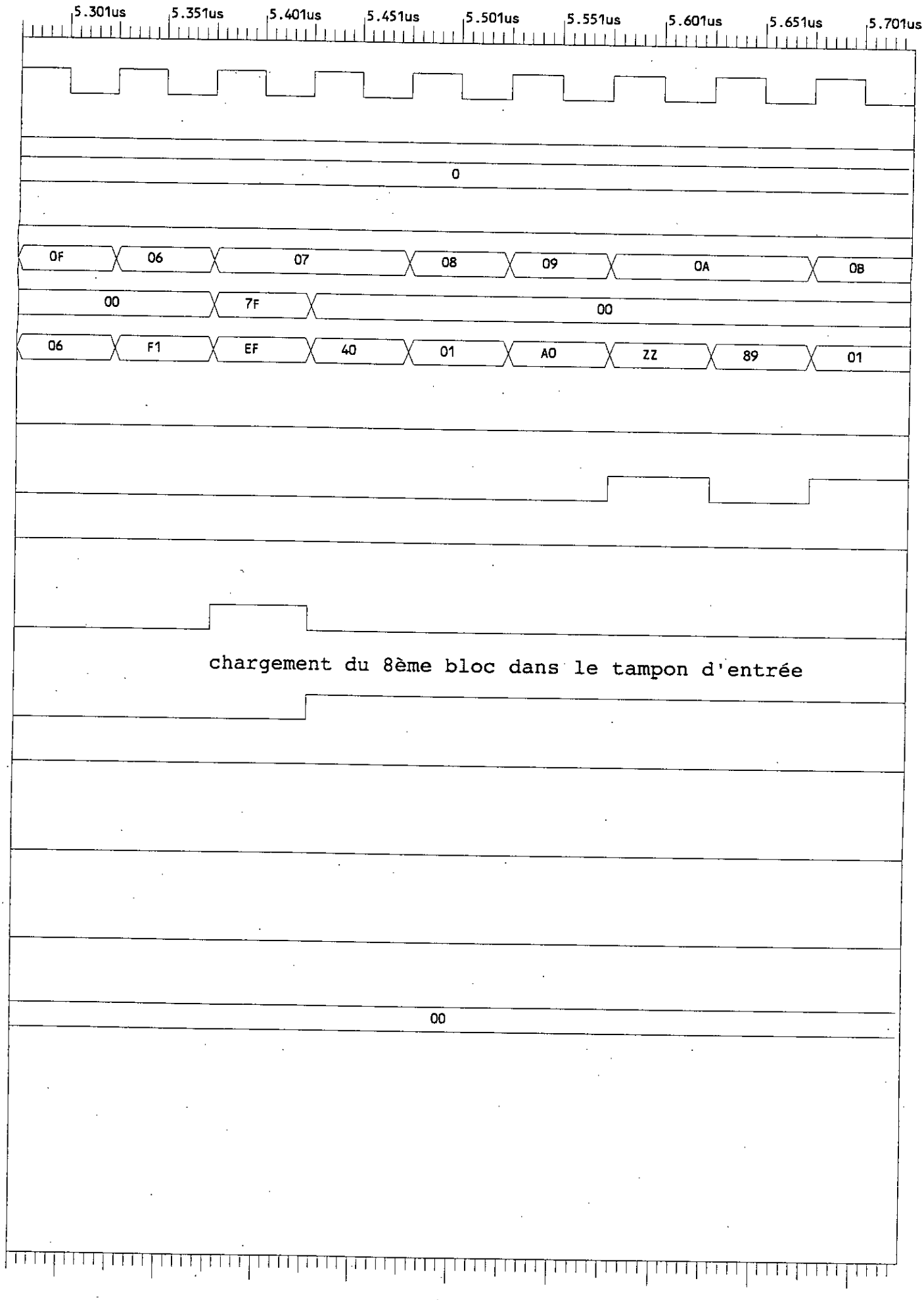



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

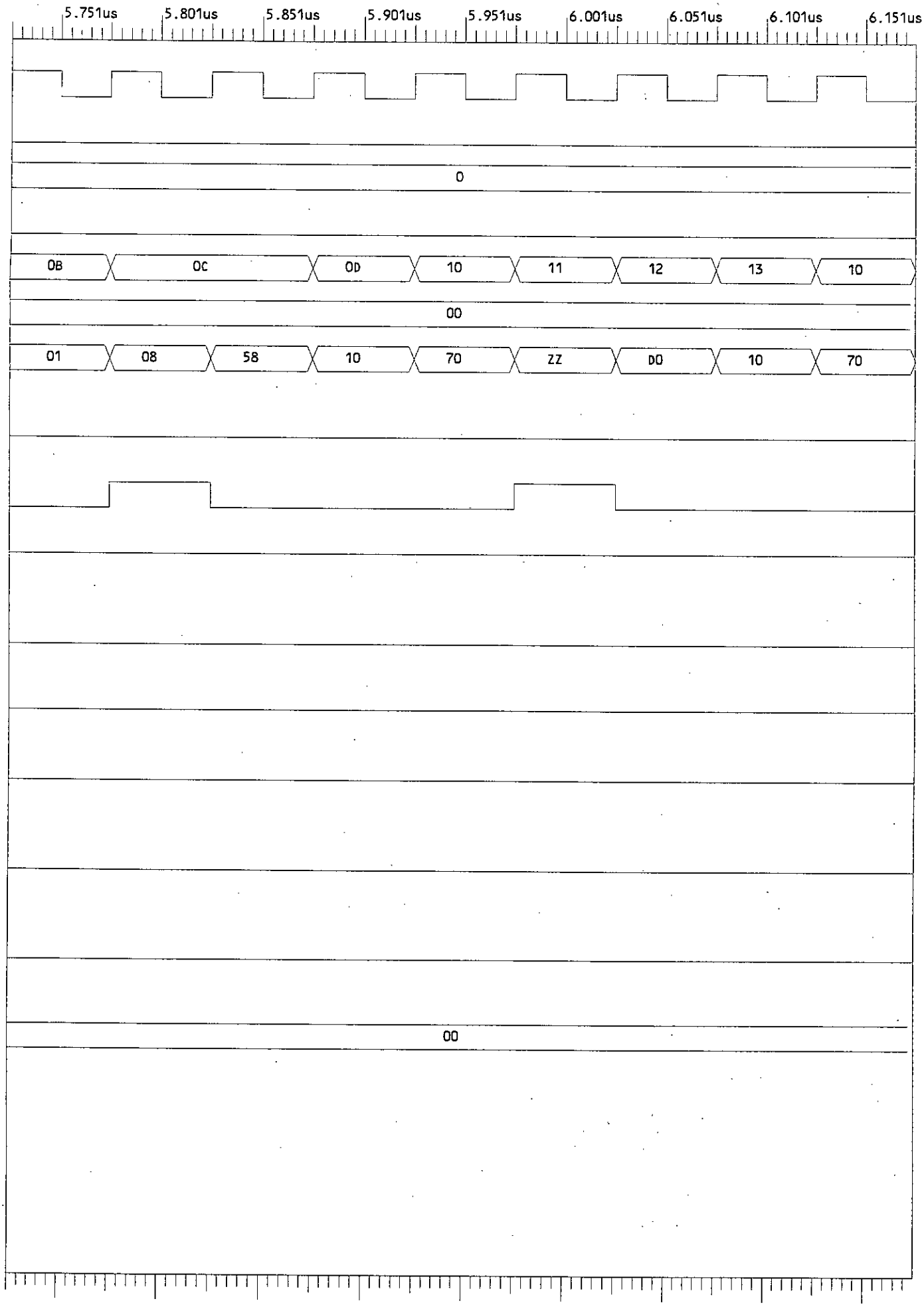



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur (suite).

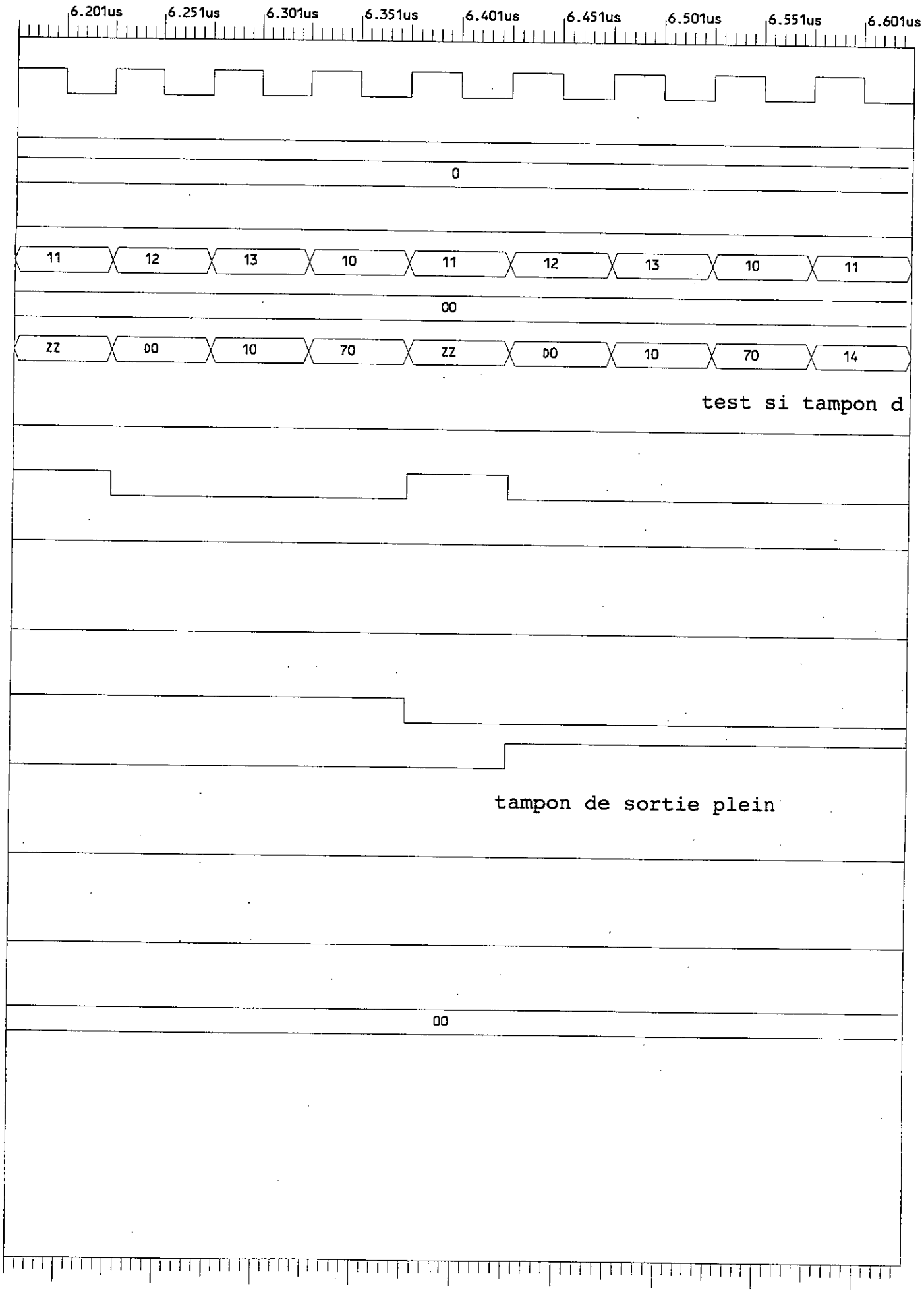



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

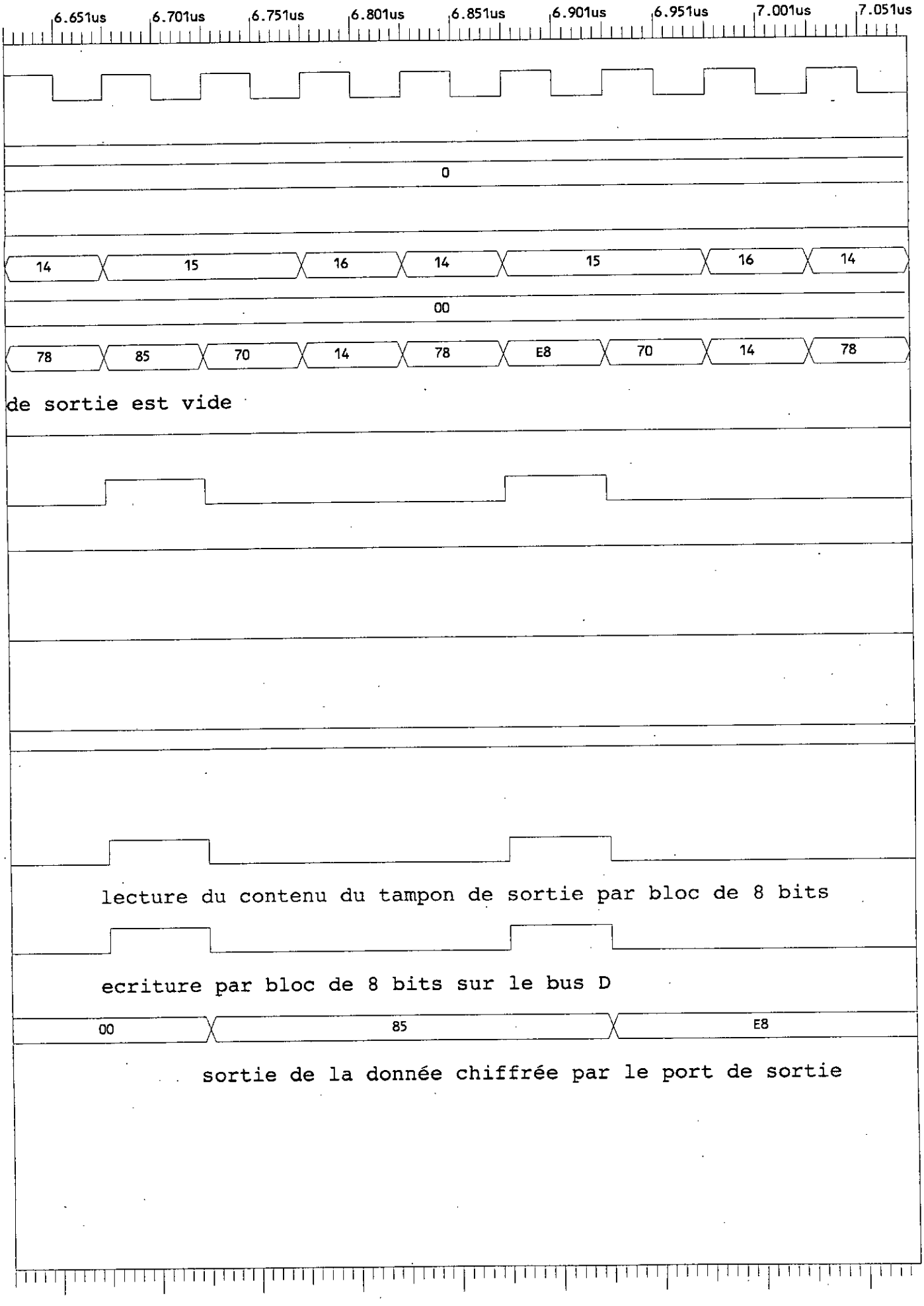



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

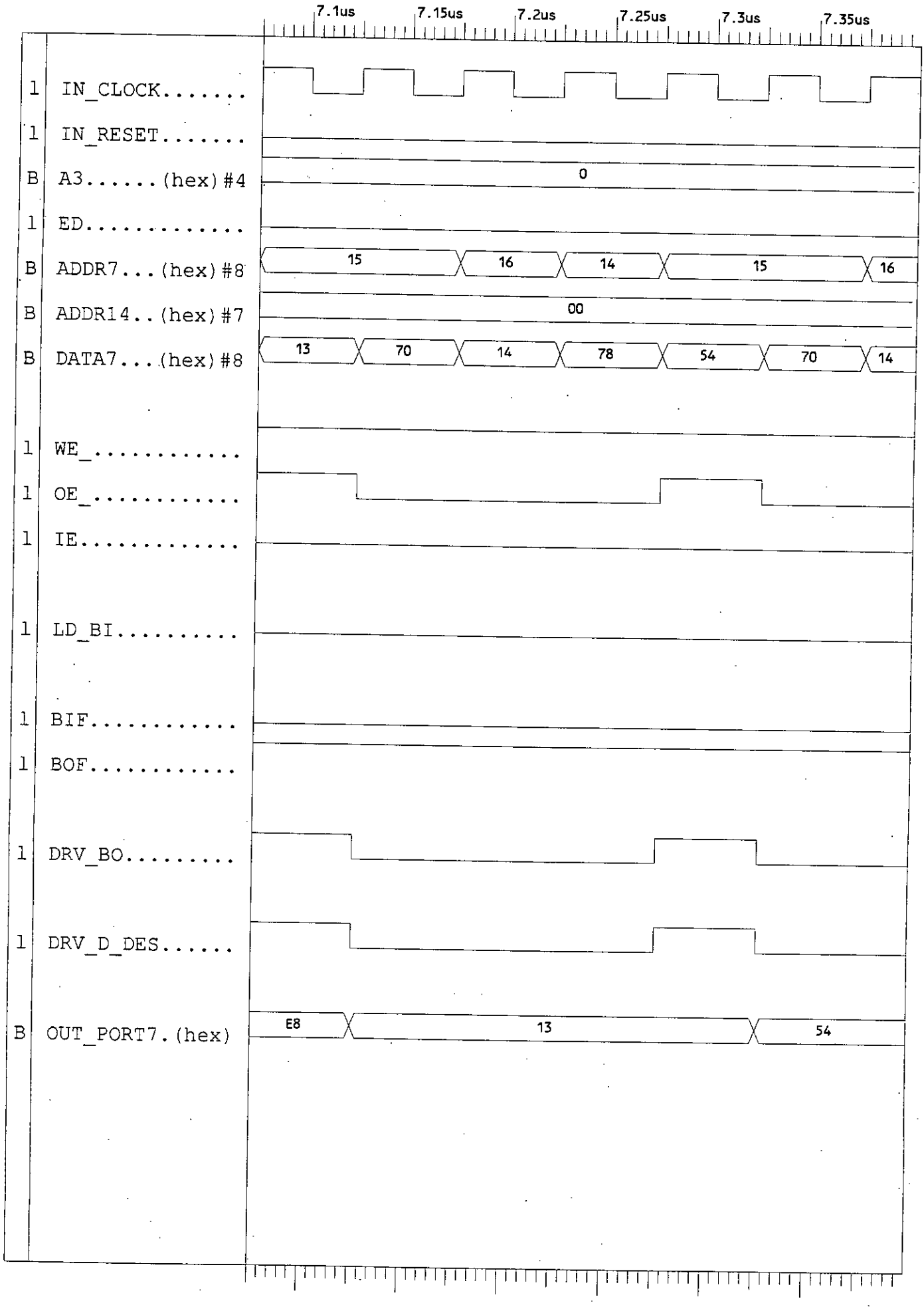



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la

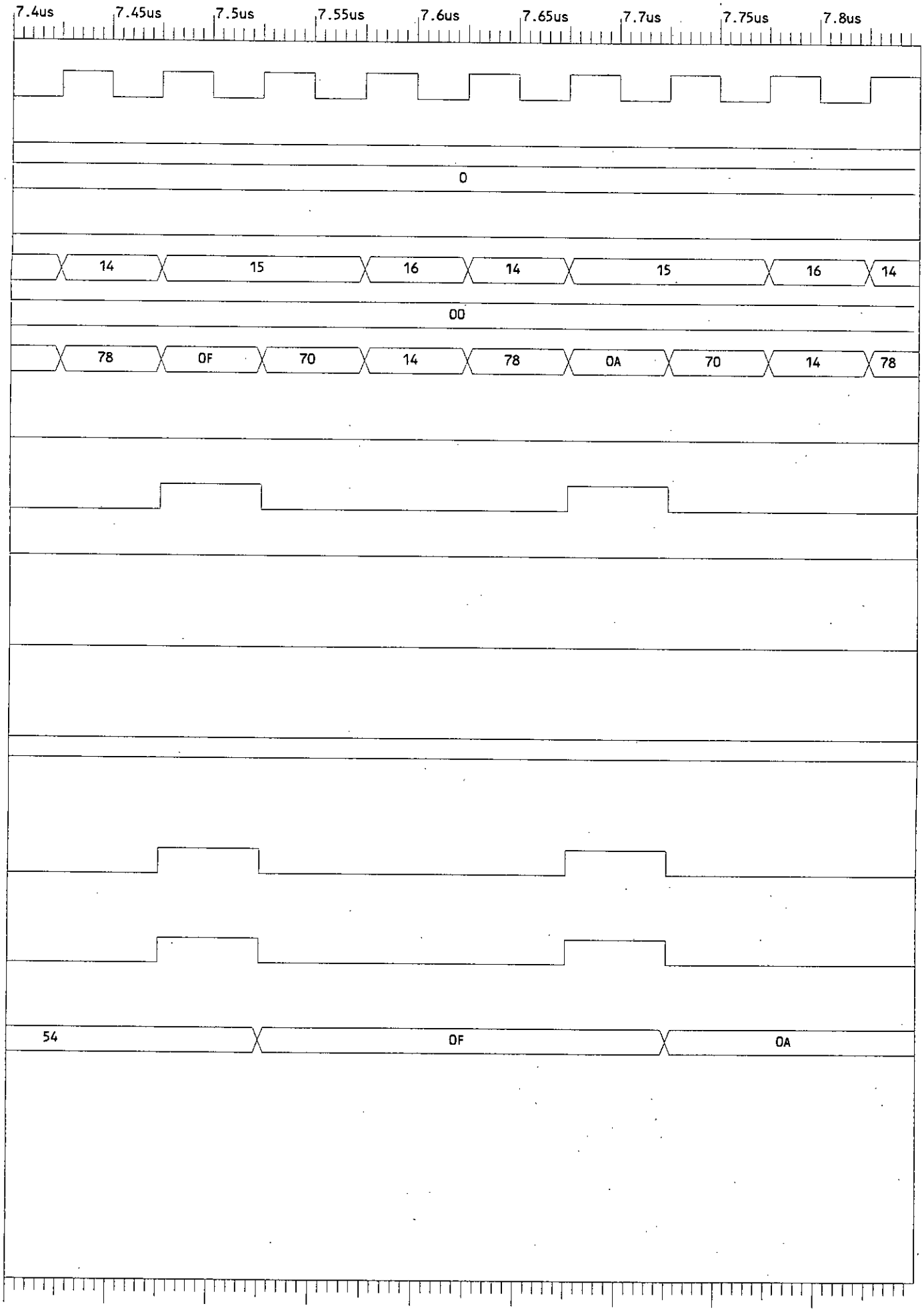



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur (suite).

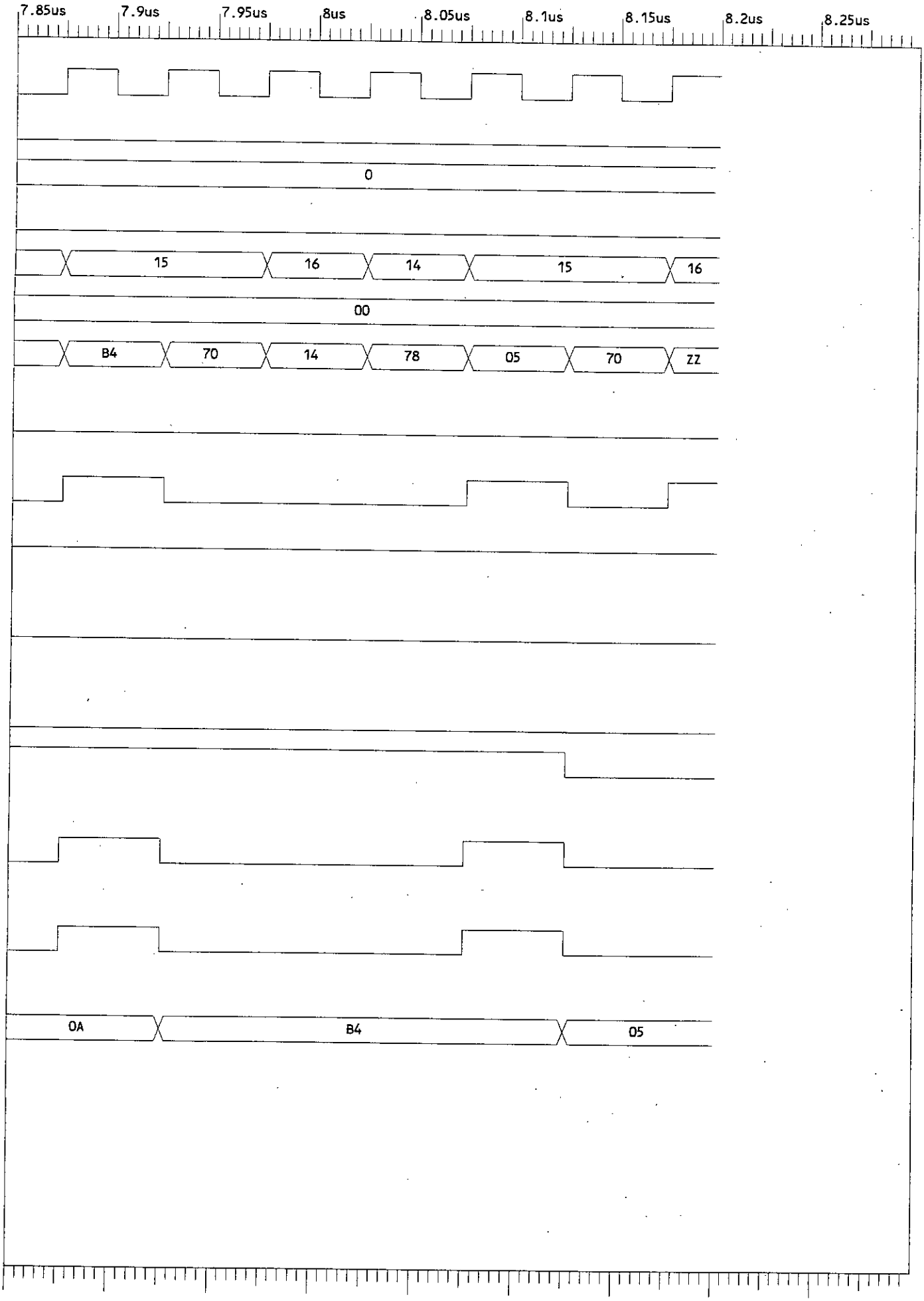



Figure E : Chiffrement d'un bloc de données de 64 bits se trouvant dans le segment DATA de la mémoire et envoi du résultat du chiffrement vers le port de sortie du cryptoprocesseur (suite).

## Annexe F

Caractéristiques du cryptoprocesseur obtenu

- **Fréquence de l'oscillateur** : 20.5 MHz
- **Durée du cycle d'horloge** : 48 ns
- **Jeu d'instructions**

  a- Instructions à usage général
  - Les instructions de transfert
  - Les instructions arithmétiques
  - Les instructions de manipulations des registres d'états
  - Les instructions d'appel et de retour de procédure
  - Les instructions d'E/S à usage général

  b- Instructions de Chiffrement/Déchiffrement
  - Les instructions de chiffrement
  - Les instructions d'initialisation et de test de l'état des tampons du bloc fonctionnel de chiffrement/déchiffrement
  - Les instructions d'E/S propre aux opérations de chiffrement/déchiffrement

- **Modes d'adressages**
  - L'adressage immédiat
  - L'adressage direct
  - adressage indexé
  - adressage indirect

- **Nombre minimum de cycle d'horloge par instruction :** 3 cycles
- **Nombre maximum de cycle d'horloge par instruction :** 5 cycles
- **Bus d'adresse :** 15 bits
- **Bus de données :** 8 bits
- **Bus de contrôle :** 3 bits
- **Entrée/sortie :** 8 bits en entrée/8 bits en sortie
- **Largeur du chemin de données interne :** 8 bits
- **Largeur du chemin du bloc de chiffrement/déchiffrement :** 64 bits
- **Nombre total de signaux externe :** 24 bits

- **Mode de fonctionnement du bloc de chiffrement/déchiffrement :** ECB "Electronic Codebook"

**ملخص**

تتناول هذه الأطروحة موضوع تصميم و غرس جهاز معالج الشفرة (cryptoprocesseur) في عنصر (FPGA) لغرض تحقيق الوظائف القاعدية للمعالج (microprocesseur) بالإضافة إلى القيام بوظيفة الشفرة و فك الشفرة التي تتحقق بواسطة خوارزمية (D.E.S).
نشرع أولا بتصميم و غرس خوارزمية (D.E.S) و مختلف بناياته في عنصر (FPGA). ثانيا نصمم و نغرس المعالج مع مزجه بوحدة الشفرة و فك الشفرة. مع إعطاء من جهة إمكانية تغيير عدة خوارزميات مرموزة لتحقيق وظيفة الشفرة و فك الشفرة لمعالج الشفرة، و من جهة أخرى نعطي إمكانية توسيع وظائف معالج الشفرة بزيادة وظائف أخرى، مثلا تغيير المفاتيح، التصديقات، و التوقيعات الرقمية.

**مفاتيح** : معالج الشفرة، معالج، حوصلة بنيوية، تصميم على عنصر FPGA، شيفرة.

**RESUME**

Ce travail consiste à concevoir et à implémenter sur un composant FPGA, un cryptoprocesseur réalisant les fonctions de base d'un microprocesseur en plus d'une fonction spécifique dédiée au chiffrement et déchiffrement des données. La fonction de chiffrement/déchiffrement du cryptoprocesseur est réalisée par l'algorithme à clef secrète D.E.S (Data Encryption Standard). En premier lieu, nous commencerons par concevoir et implémenter sur un composant FPGA l'algorithme D.E.S et de ses divers options architecturales. En second lieu, nous procéderons à la conception et l'implémentation du microprocesseur couplé à l'unité de chiffrement/déchiffrement, tout en donnant d'une part la possibilité d'interchanger plusieurs algorithmes cryptographiques pour la réalisation de la fonction de chiffrement/déchiffrement du cryptoprocesseur et d'autre part la possibilité d'étendre les fonctionnalités du cryptoprocesseur par l'ajout d'autres fonctions cryptographiques, à savoir, échange de clefs, authentification et signature numérique.

**Mots clés** : Cryptoprocesseur, Algorithme DES, Synthèse architecturale, Conception sur composants FPGAs, Cryptographie.

**ABSTRACT**

This work consists to conceive and to implement on a FPGA component, a cryptoprocessor achieving the basics functions of a microprocessor in addition to a specific function dedicated to ciphering and deciphering data. The function of ciphering/deciphering of the cryptoprocessor is achieved by a secret key algorithm D.E.S (Data Encryption Standard). First, we will start with conceiving and implementing on a FPGA component the algorithm DES and its various architectural options. In second, we will proceed to conceive and to implement the microprocessor coupled to the unit of ciphering and deciphering data. While giving the possibility to switch several cryptographic algorithms in order to realise the ciphering/deciphering function of the cryptoprocessor and on the other hand the possibility to spread functionalities of the cryptoprocessor by the addition of other cryptographic functions, such as, keys exchange, authentification and digital signature.

**Key words** : Cryptoprocessor, Algorithm DES, Algorithmic synthesis, Conception on FPGAs components, Cryptography.