République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Abou Bakr Belkaid -Tlemcen.
Faculté de Technologie.
Département de Génie Electrique et Electronique.

Mémoire De Magister En

Micro-électronique

THEME

Conception d'un contrôleur vidéo en technologie FPGA

Application d'un jeu vidéo

Présentée par **: Mme BABA HAMED Amel née BENSEMAIN**

Soutenue devant le jury :

Mr K.E GHAFFOUR	Professeur à l'université Abou Bakr Belkaid Tlemcen.	..Président du jury.
Mr M.A CHIKH	Maître de conférences à l'université Abou Bakr Belkaid. Tlemcen.	…….Examinateur.
Mr S.M KARABERNOU	Maître de conférences à L'ENSEA. Paris, France.	…….Examinateur.
Mr A.H BESSAID	Professeur à l'université Abou Bakr Belkaid. Tlemcen.	…….Encadreur.
Mr H BECHAR	Chargé de cours à l'université Abou Bakr Belkaid. Tlemcen.	…...Co-encadreur.

Année Universitaire : 2009-2010.

Dédicaces

A mes très chers parents.

Pour leur amours, leur soutien.

A mon très cher époux Amine.

Il été toujours là à mes cotés.

A mes très chers et adorables enfants :

Djihane et Imade-Eddine.

A ma très chère et unique sœur Soraya et sa petite famille.

A mes très cher frères : Nabil et Toufik,

et à Djallel et sa petite famille.

A mes amies : Hanane, Nadjet, Samia, Nawel, Ilhem, Hassiba, Imane,

Lamia, Amel, Sonia, Souade.

A tous mes collègues de la Faculté de Technologie

en particulier les laborantins du département de Génie Electrique.

Amel

Remerciements

Je remercie ALLAH tous puissant, qui m'a donné la force, la patience et le courage afin de concilier entre ma vie familiale, professionnelle et mes études.

Je témoigne ma gratitude à mon encadreur Monsieur A.H BESSAID, professeur à l'université Abou Bakr Belkaid de Tlemcen, pour son encadrement, son soutien, ses précieux conseils et ses encouragements.

Mes très vifs remerciements accompagnés de toute ma gratitude vont à mon co-encadreur Monsieur H BECHAR, chargé de cours à l'université Abou Bakr Belkaid de Tlemcen, pour avoir contribué activement à la réalisation de ce projet, pour ses conseils efficaces, pour ses encouragements et pour ses grandes qualités humaines.

Je remercie Monsieur K.E GHAFFOUR, Professeur à l'université Abou Bakr Belkaid de Tlemcen, pour l'honneur qu'il m'a fait en acceptant de présider le jury de ce mémoire.

Je remercie Monsieur M.A CHIKH, maître de conférences à l'université Abou Bakr Belkaid de Tlemcen, pour l'intérêt qu'il a porté à mes travaux en acceptant d'examiner ce mémoire.

Je tiens à exprimer ma reconnaissance à Monsieur S.M KARABERNOU, maître de conférences à l'Ecole Nationale Supérieure de l'Electronique et de ses Applications (ENSEA) de Cergy Pontoise, Paris, qui m'a encadré pendant mon stage à l'ENSEA. Je le remercie pour ses conseils et ses encouragements. Il m'a fait un grand honneur en acceptant d'examiner ce travail.

Ces quelques lignes ne pourront jamais exprimer la reconnaissance que j'éprouve envers tous ceux qui, de près ou de loin, ont contribué par leurs conseils, leurs aides et leurs encouragements pour l'aboutissement de ce travail.

Table des matières

Table des illustrations

Liste des figures

Liste des tableaux

Introduction générale

Introduction :

Au cours des vingt dernières années, les méthodes de conception des fonctions numériques ont subi une évolution importante. Dans les années soixante, la majorité des applications de la logique câblée étaient construites autour de circuits intégrés standard, souvent pris dans la famille TTL, et dans les années soixante dix, les premiers circuits programmables par l'utilisateur ont été réalisé : les SPLD (*Simple Programmable Logic Devices*). Au début des années quatre vingt, apparurent, parallèlement, les circuits intégrés spécifiques pour les fonctions complexes fabriquées en grande série ASIC (*Application Specific Integrated Circuits*) et les FPGA (*Field Programmable Gate Array*) de quelques dizaines de milliers de portes équivalentes.

La conception des circuits numériques spécialisés est actuellement au cœur d'enjeux économiques extrêmement importants, liés à l'explosion du marché des applications dans différents domaines (le prototypage, le cryptage de données, …). La mise en œuvre matérielle de ces applications doit intégrer de nombreuses contraintes, portant à la fois sur le coût, les performances, et la consommation électrique des circuits réalisés. La réduction des risques économiques liés à la mise sur le marché d'un nouveau produit impose donc, d'une part, de limiter ses coûts de conception, et d'autre part de minimiser le temps écoulé entre sa spécification et sa commercialisation.

Parallèlement, les progrès réalisés en micro-électronique permettent aujourd'hui d'intégrer plusieurs centaines de millions de transistors sur un seul circuit, cette capacité d'intégration augmentant exponentiellement selon la loi de Moore. Ces évolutions technologiques ont particulièrement favorisé la famille des circuits programmables FPGA (*Field Programmable Gate Array*), qui sortent aujourd'hui de leur niche de marché originale pour se poser comme une alternative aux circuits dédiés ASIC (*Application Specific Integrated Circuit*). L'intérêt suscité par les FPGA est dû essentiellement à leurs prix abordables, facilité de mise en œuvre et flexibilité. En outre, les coûts fixes et délais de fabrications, en comparaison avec les circuits spécifiques (ASIC), sont totalement éliminés.

Objectifs :

C'est dans le contexte général dressé précédemment que se situe le travail de ce mémoire. Dans la technologie microélectronique, la conception d'un circuit numérique VLSI (*Very Large Scale Integration*) se fait principalement via deux approches : la première est utilisée par les techniques de conception rapide telle que celle des circuits programmables (FPGA), la seconde approche consiste à réaliser des circuits beaucoup plus optimisés et plus performants (ASIC).

Les inconvénients majeurs d'une conception optimisée résident dans le fait du passage obligatoire chez le fondeur, ce qui implique des frais de développement élevés du circuit, et il s'avère nécessaire d'être rigoureux lors de la phase de développement, de telle sorte que le

circuit prototype fonctionne dès les premiers essais. Pour cela, nous avons choisi d'établir la première approche, qui permet de concevoir et réaliser des architectures avec relativement un faible investissement de développement.

La démarche que nous avons suivie tout au long de notre travail comporte deux points principaux:

- ✓ Une recherche bibliographique qui nous a permis d'établir l'état de l'art dans le domaine des composants programmables, en particulier les FPGA.

- ✓ Une étude et un apprentissage approfondi sur les langages de description de haut niveau et sur les outils de CAO (Conception Assistée par Ordinateur), en particulier le VHDL (*Very High Speed Integrated Circuit, Hardware Description Language*) **[WEBER-07]** et le ISE 9.2i de XILINX. Nous avons commencé notre travail par des conceptions simples (portes de base, décodeurs, multiplexeurs,…) jusqu'à des conceptions plus étendues comprenant un contrôleur de VGA.

Notre projet de conception matérielle est l'implémentation d'un circuit complet sur une carte de développement FPGA (NEXYS2) de DIGILENT comprenant un SPARTAN 3E de XILINX.
Cette architecture matérielle consiste à afficher sur le moniteur VGA un jeu vidéo, comprenant :

- une raquette qui doit se déplacer horizontalement en bas d'écran sous l'action du joueur.
- une balle qui se déplace selon une trajectoire de 45° avec la verticale et qui rebondisse lorsqu'elle touche l'un des trois bords de l'écran (gauche, droit et haut) ou la raquette.
- Le jeu sera terminé lorsque la balle touche le bord bas de l'écran.

Le travail principal consiste à répartir de façon judicieuse l'ensemble du circuit dans les différents éléments du composant programmable FPGA, en exploitant au maximum les possibilités du langage VHDL pour une réalisation parfaite du projet.

État de l'art :

Depuis la commercialisation du premier circuit programmable FPGA en 1985, l'utilisation de ce dernier ne cesse de s'étendre à des domaines et applications variés, parmi lesquels nous pouvons citer : le prototypage rapide **[MOSHER-10],** le traitement d'image **[GHOZZI-03], [SEDCOLE-06]**, le cryptage des données **[CHODOWIEC-03]**, le calcul reconfigurable **[RADUNOVIC-99], [DIESSEL-99]**, l'estimation de mouvement **[WOODFILL-97], [KARABERNOU-05],** les processeurs programmables DSP **[DELAHAYE -04],** le mode de reconfigurabilité **[DJUPDAL-05], [GUERMOUD-97], [ABEL-06],** les systèmes embarqués **[MRABET-05]**, **[GHALI -05]**, …etc.

A cause de leur coût réduit ainsi que de leur souplesse de programmation, les FPGA sont aussi largement utilisés dans le prototypage rapide. Par mesure de précaution, avant d'entamer la gravure d'un circuit intégré, celui-ci est toujours soigneusement testé par simulation de manière à garantir la correction. L'utilisation de circuits FPGA permet ainsi d'accélérer grandement cette simulation par rapport à une simulation logicielle **[MOSHER-10]**.

Le traitement d'images en temps réel nécessite l'utilisation des circuits électroniques rapides, capables de manipuler de grandes quantités d'informations générées par la source vidéo. Pour cela, les FPGA sont particulièrement adaptés **[GHOZZI-03], [SEDCOLE-06]**.

Les cellules du FPGA étant indépendantes les unes des autres, elles peuvent très bien effectuer leurs calculs en parallèle, ce qui les rendent rapide permettent d'intégrer différents algorithmes.

- ✓ La protection des données numériques devient importante pour de nombreuses raisons telles que la confidentialité et l'intégrité. Le cryptage de ces données répond à ce problème de confidentialité. Donc le principe est d'implanter des algorithmes de cryptage dans un circuit reconfigurable FPGA afin de permettre le chiffrement/déchiffrement de blocs à l'aide des clés pour un niveau de sécurité acceptable. On trouve comme des algorithmes de cryptage tel que DES (*Data Encryption Standard*), AES (*Advanced Encryption Standard*) **[CHODOWIEC-03].**

- ✓ De nombreux algorithmes nécessitent la multiplication. L'architecture du composant programmable contient des LUT (*Look-Up Tables*) qui simplifient la multiplication. Ces tables contiennent des valeurs pré-calculées, éliminant ainsi l'ajout d'une cellule multiplieur **[RADUNOVIC-99]**.

- ✓ Les circuits programmables fournissent une puissance de calcul importante. Des algorithmes, traditionnellement considérés comme une séquence d'instructions, peuvent être analysés pour déterminer un parallélisme possible. La possibilité de reconfiguration sur ces circuits permet une minimisation et une adaptation spécifique à chaque algorithme. **[DIESSEL-99].**

- ✓ Toujours, dans l'implantation des algorithmes dans un circuit reconfigurable FPGA, on trouve la transformée de Census, qui permet d'estimer le mouvement de chaque pixel entre deux images acquises successivement. Il est basé sur un calcul de luminosité entre pixels proches **[WOODFILL-97].**

- ✓ On a aussi une implémentation en temps réel dans un FPGA de la transformée de Hough en employant le gradient et l'algorithme de CORDIC pour la détection de lignes droites. L'algorithme de CORDIC permet l'utilisation des opérateurs simples, tels que des additionneurs et des registres à décalage, tandis que l'utilisation des

gradients réduit considérablement la quantité de calcul. Cette approche peut être facilement appliquée pour la détection d'autres formes **[KARABERNOU-05].**

L'utilisation des FPGA pour l'implantation des algorithmes de contrôle de courant est une alternative à l'emploi de DSP. En effet, pour une réalisation matérielle de l'algorithme, les FPGA permettent d'exploiter toutes les possibilités de parallélisme d'un algorithme donné. En implantant concurremment toutes les tâches qui peuvent être exécutées simultanément, le temps de traitement peut être réduit dans des proportions importantes.

- ✓ On trouve dans **[DELAHAYE -04]** la possibilité d'intégrer sur une plate-forme un DSP et FPGA (plate-forme hétérogène) au lieu d'une plate-forme multi-DSP, afin de réaliser la reconfiguration dynamique et partielle de la plate-forme**.**

Plusieurs systèmes à base de FPGA sont décrits dans la littérature. Nous pouvons les distinguer par leurs modes de configuration. Les premiers utilisent un mode de configuration statique, pour lequel nous avons une configuration du système par application (reconfiguration entre les applications), ex: **[DJUPDAL-05]**. Les seconds utilisent un mode dynamique consistant à reconfigurer le système plusieurs fois durant l'exécution de la même application **[GUERMOUD-97], [ABEL-06]**.

- ✓ Certains FPGA, comme ceux conçus par ACTEL par exemple **[Xil.com]**, sont programmés de manière irréversible grâce à des anti-fusibles, d'autres reposent sur une technologie SRAM (*Static Random Access Memory*), comme ceux proposés par XILINX **[Xil.com]** ou ALTERA **[Alt.com]**. Cette technologie permet ainsi de reconfigurer le FPGA autant de fois que nécessaire. Cela est donc utile lorsqu'une erreur est découverte dans un circuit car il suffit alors de reprogrammer le FPGA avec une version corrigée du circuit **[DJUPDAL-05]**.

Le principe de la reconfiguration dynamique consiste à reconfigurer un ensemble de FPGA-SRAM pendant l'exécution d'une même application. Cette dernière est décomposée au préalable en plusieurs étapes successives. Chaque reconfiguration correspond alors à l'exécution d'une étape de l'application.

- ✓ **[GUERMOUD-97]** présente l'intérêt d'utiliser la reconfiguration dynamique des FPGA pour le traitement d'image, l'architecture proposée permet de chaîner deux opérateurs (filtre médian et gradient de sobel) sur une image.

- ✓ Dans l'approche de **[ABEL-06],** l'étude s'effectue sur les différentes architectures reconfigurables à base de FPGA afin de concevoir une architecture générique permettant, en utilisant la reconfiguration dynamique, de s'adapter à un nombre important d'algorithmes.

On assiste ainsi à l'émergence de nouvelles générations de circuits programmables, qui offrent des densités de plusieurs dizaines de millions de portes logiques, et intègrent des

cœurs de processeurs hautes performance ou des blocs de propriété intellectuelle IP qui sont des modules sous forme code VHDL déjà synthétisés et compilés et intégrés dans une bibliothèque, réutilisés en cas de besoin ultérieur pour construire d'autres circuits plus complexes.

- ✓ L'objectif est de fournir aux concepteurs de systèmes des « plates-formes de conception » reconfigurables, qui devraient permettre la mise en œuvre de SOC (*System On a Chip*) à base de logique programmable **[MRABET-05]**.

- ✓ L'avènement des dernières générations de FPGA et la disponibilité des blocs de propriété intellectuelle IP (*Intellectual Property*), utilisable notamment sur ces matrices programmables, mettent la technologie SOC à la portée d'un public nettement plus large **[GHALI -05]**. Ainsi, cette technologie, qui ciblait principalement les domaines : public, médical, télécommunication et automobile, devient intéressante pour des applications plus simples et portant sur des volumes moins importants.

Plan du mémoire :

La suite de ce mémoire est organisée de la façon suivante :

Chapitre 1 : Entièrement consacré à l'étude des composants programmables, il commence par un historique des différents composants : les circuits standards, les circuits spécifiques ASIC (*Application Specific Integrated Circuit*), les circuits programmables qui combinent les composants programmables simple SPLD (*Simple Programmable Logic Devices*), les CPLD (*Complex Programmable Logic Devices*) et les réseaux programmables FPGA (*Field Programmable Gate Array*). Pour ce dernier, qui est le plus développé actuellement, nous avons décrit ses différentes architectures et ses différents éléments (blocs logiques, les entrées/sorties, les interconnexions). Nous avons aussi présenté les technologies d'interconnexions utilisées dans la logique programmable, enfin nous avons dévoilé les critères de choix entre les différents circuits numériques.

Chapitre 2 : Ce chapitre expose les méthodologies de conception des circuits numériques, premièrement nous présentons les méthodes de conception en détaillant la plus utilisée pour les circuits VLSI (*Very Large Scale Integration*) : ASIC, CPLD et FPGA. Nous abordons ensuite les différents outils de développement (les outils de CAO et les diverses descriptions d'un circuit ainsi que les langages de conception), en nous intéressant en particulier à la synthèse matérielle de haut niveau utilisées pour les circuits de hautes densités.

Chapitre 3 : Ce chapitre constitue le cœur de ce mémoire. L'architecture matérielle que nous proposons y est exposée. Nous commençons par y décrire l'environnement de travail de notre application en présentant les différents outils de développement (le logiciel ISE 9.2i de XILINX et la carte NEXYS2 de DIGILENT). Dans la deuxième partie de ce chapitre, nous avons exposé notre description matérielle du projet en passant par tous les étapes essentielles d'une réalisation architecturelle. Pour notre projet, nous avons réalisé deux conceptions

différentes dont nous détaillons les blocs utilisés (leurs descriptions fonctionnelles ainsi que leurs structures internes). Ce chapitre se termine par une implémentation des deux conceptions dans le circuit reconfigurable FPGA.

Nous terminons enfin cette réalisation par une **conclusion** qui résume les travaux effectués dans le cadre de ce projet, et qui offre des orientations pour des travaux futurs.

Trois **annexes** regroupent différentes informations nécessaires au déroulement de notre projet, nous trouvons dans la première annexe la procédure qui définit l'utilisation des outils de CAO, la seconde le schéma synoptique de la carte de développement ainsi que les descriptions schématiques de chaque élément. Enfin, la dernière annexe est consacrée à l'outil ADEPT (propriétaire de DIGILENT) qui permet la configuration du composant programmable FPGA.

Chapitre 1

Les techniques d'intégration des systèmes numériques

1.1 Introduction :

Depuis longtemps, le besoin de miniaturisation et de réduction des coûts ont poussé les fabricants de composants électroniques à intégrer au maximum les différentes structures donnant naissance aux circuits intégrés. Ces derniers sont des composants électroniques reproduisant une ou plusieurs fonctions électroniques plus ou moins complexes, intégrant plusieurs types de composants de base dans un volume réduit. Ce circuit est appelé puce électronique.

En micro-électronique, le matériau de base le plus couramment utilisé pour la réalisation des circuits intégrés se présente sous forme de plaquettes de silicium monocristallin. Ces plaquettes (*wafers*) circulaires sont de différents diamètres et d'une épaisseur qui vari de 600 à 700µm (Figure 1.1), on peut dire que le wafer est un disque assez fin de matériau qui sert de support à la fabrication de microstructures par des techniques telle que le dopage, la gravure, la déposition d'autres matériaux (épitaxie, dépôt chimique…) et la photolithographie.

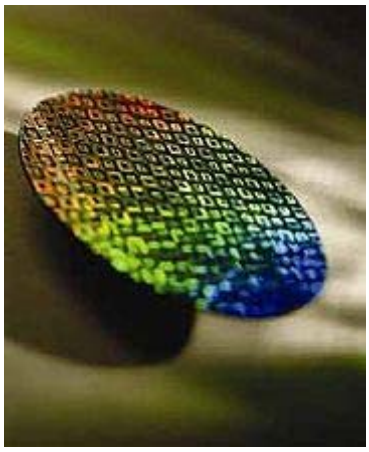


Figure 1.1 : Un wafer en silicium gravé.

La fabrication consiste d'abord à créer sur le cristal de silicium, appelé substrat, les zones actives des transistors (Drain, Source, Grille s'il s'agit de MOS) selon une géométrie précise de façon à optimiser l'encombrement tout en respectant les caractéristiques électriques de fonctionnement, puis à interconnecter ces transistors par des liaisons métalliques. Ces opérations se font chez un industriel appelé "fondeur" (comme : MATRA MHS, SGS THOMSON ...) qui dispose de la description topologique de chacun des niveaux qu'il traite (diffusion N ou P pour le drain et la source, polysilicium pour la grille, métal pour les liaisons). C'est grâce à cette description qu'il fabrique des masques (éléments de base pour la fabrication des circuits intégrés).

A partir de 1947, l'invention des transistors a conduit au développement de circuits particuliers, tout circuit étant nouveau. Puis ont été développés les composants discrets standards, permettant par assemblage de ces composants la réalisation de circuits ou de systèmes spécifiques. La période suivante a vu l'émergence des premiers microprocesseurs et le développement des mémoires statiques et dynamiques, conduisant aux micro-ordinateurs et

microcontrôleurs et mettant l'accent sur la programmation des applications utilisant cette approche. Parallèlement, les premières réalisations matérielles des circuits programmables apparaissent sur le marché aux années 70. L'apparition de ce type de circuit a d'abord commencée par les circuits logiques programmables simples de type PAL (*Programmable Array Logic*), qui sont utilisés pour implémenter des fonctions combinatoires simples, et ils se programment comme des mémoires non volatiles de type PROM.

Avec les évolutions en micro-électronique et l'augmentation de la densité d'intégration, différentes familles de circuits intégrés ont commencé à apparaître : les ASIC (*Application Specific Integration Circuit*), les CPLD (*Complex Logic Programmable Device*), puis les FPGA (*Field Programmable Gate Arrays*), introduits par la société XILINX en 1985 **[DERRIEN-02]**. A l'heure actuelle, on compte une dizaine de fabricants, le marché étant nettement dominé par les sociétés XILINX, ALTERA (circuits reprogrammables) et ACTEL (circuits non reprogrammables).

Dans ce chapitre, nous présenteront les différents circuits numériques, nous insisterons sur les circuits logiques programmables, et en particulier sur l'architecture interne des circuits reconfigurables FPGA, car notre travail s'est effectué sur ce dernier type.
Nous présenterons aussi les différentes technologies d'interconnexions utilisées dans les circuits programmables ainsi que les critères de choix entre les circuits numériques.

1.2 Architectures classiques des circuits numériques :

Le rôle de la technologie micro-électronique est la réalisation et l'intégration des transistors nécessaires à la réalisation des circuits électronique. Cette technologie d'intégration se présente sous formes de plusieurs possibilités (Figure 1.2):

- Utiliser des circuits standards, en les associant sur un ou plusieurs circuits imprimés pour réaliser le système désiré.
- Concevoir des circuits spécifiques, ce qui permet d'avoir une bonne densité d'intégration.
- Utiliser des circuits programmables qui peuvent être une solution intermédiaire entre les deux précédents.

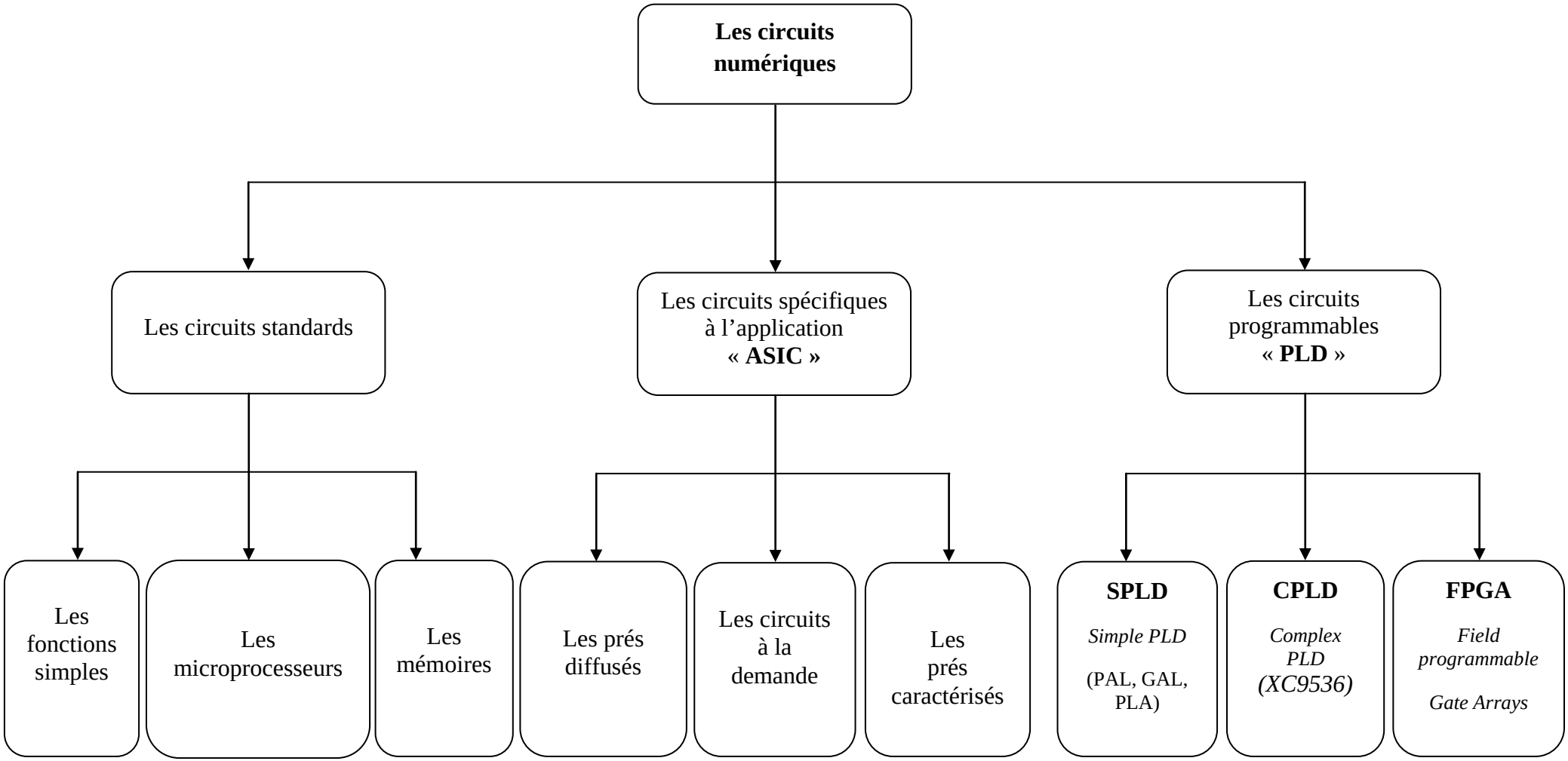


Figure 1.2 : Classification des circuits numériques.

1.2.1 les circuits standards :

Des fabricants de circuits tels que MATRA, MOTOROLA, SGS THOMSON, … proposent des composants standards ayant des fonctions plus ou moins complexes. L'association de ces composants sur un circuit imprimé permet de réaliser un système numérique.

L'avantage de cette technique réside dans le fait que la réalisation du système peut être relativement rapide. Cependant les composants n'étant pas exactement adaptés aux besoins de l'application, le taux d'intégration (c'est à dire la complexité du système par rapport à la quantité de composants utilisés) est, en général, relativement modeste. C'est une technique bien adaptée aux petites séries. On peut classer les circuits standards comme suit :

1.2.1.1 Les fonctions simples :

Certains circuits combinatoires de moyenne complexité MSI (*Medium Scale Integration*) sont considérés comme des circuits standards de base. Les fonctions correspondantes réalisées en circuits intégrés se retrouvent comme composants dans les bibliothèques (librairies) de conception des circuits logique programmables **[NKETSA]**. On peut trouver aussi les circuits SSI (*Small Scale Integration*), qui réalisent des fonctions combinatoires ou séquentielles élémentaires.

1.2.1.2 Les microprocesseurs :

Pour minimiser l'impact du coût de conception et de fabrication des circuits intégrés les plus complexes, il est intéressant de leur donner une gamme d'applications permettant de s'adresser à une clientèle la plus large possible. Dans cet esprit, il s'agit de créer un circuit de traitement numérique dont l'usage final (l'application) n'est pas connu à la fabrication. Pour cela, il suffit de réaliser un circuit intégré ayant quelques ressources de traitement assez génériques (addition de deux nombres, stockage d'un nombre en mémoire, lecture d'un nombre d'une mémoire...) associées à un dispositif de contrôle capable d'interpréter des ordres simples qui sont stockés dans une mémoire extérieure au circuit. Il suffit de changer le contenu de cette mémoire (le programme) pour changer l'enchaînement des traitements effectués par le circuit, donc l'application. Avec de tels circuits l'augmentation de complexité des applications est gérée simplement par l'augmentation de la taille des programmes. On trouve comme circuits les processeurs à usage général, les processeurs spécialisés qui sont semblables aux derniers mais ils proposent un jeu d'instructions liés à une architecture adaptée à une famille d'algorithmes (par exemple : processeur de traitement de signal DSP (*Digital Signal Processor*) **[Wikp.com]**.

1.2.1.3 Les mémoires :

Une mémoire est un dispositif permettant de stocker puis de restituer une information. On distingue deux classes de mémoires à semi conducteur :

- Les mémoires vives : sont des mémoires volatiles, car on peut perdre l'information en cas de coupure d'alimentation électrique. Elles peuvent être lues et écrites.

- Les mémoires mortes : sont des mémoires qui conservent l'information même en absence de l'alimentation. Donc on peut les considérées comme un circuit logique programmable.

1.2.2 les circuits spécifiques à l'application ASIC (*Application Specific Integration circuit*) :

Les circuits ASIC constituent la troisième génération des circuits intégrés qui a vu le jour au début des années 80. En comparaison aux circuits intégrés standards et figés proposés par les fabricants, l'ASIC présente une personnalisation de son fonctionnement, selon l'utilisation, accompagnée d'une réduction du temps de développement, d'une augmentation de la densité d'intégration et de la vitesse de fonctionnement **[TARIGHT-99]**.

On peut dire que l'ASIC est un circuit intégré qui permet un câblage direct des applications spécifiques sur le silicium. Par exemple, une opération cruciale en télévision numérique, l'estimation de mouvement est réalisée actuellement par un seul circuit ASIC capable de calculer plus de dix milliards d'additions par seconde, ce qui dépasse de loin les performances des meilleurs microprocesseurs. La contrepartie à cette performance est que ce circuit ne peut servir qu'à la télévision numérique...

1.2.2.1 Les prés diffusés (*Gate Array*):

Les prés diffusés sont des circuits qui regroupent tous les éléments électroniques (transistors, diodes, résistances, capacités, etc.) suivant une certaine topologie, sans aucune connexion entre eux (sauf au niveau diffusion) ce qui nous ramène à dire que les réseaux pré diffusés sont des circuits partiellement préfabriqués.
La réalisation des connexions dans le but de définir la fonction souhaitée est la tâche du concepteur, pour cela, il dispose de bibliothèques de macro-cellules et d'outils logiciels d'aide à la conception. A partir de cette liste d'interconnexions (*netlist*), le fondeur aura quelques étapes technologiques à effectuer pour achever le circuit (Figure 1.3), c'est à dire le dépôt d'une ou plusieurs couches de métallisation **[TARIGHT-99]**.

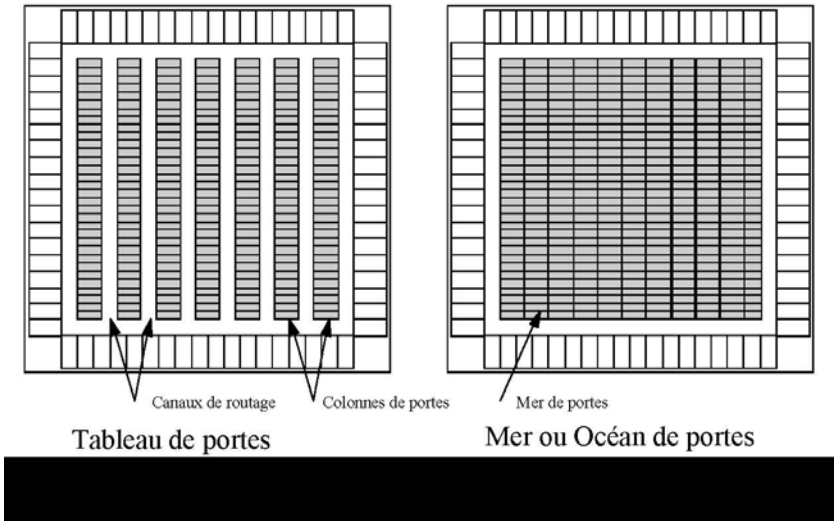


Figure 1.3 : Les circuits prés diffusés.

1.2.2.2 Les circuits à la demande (*Full-Custom*):

Ce sont des circuits où chaque transistor est optimisé individuellement. On dispose d'une bibliothèque de modèles mathématiques de comportement et via un "compilateur de silicium", logiciel très sophistiqué, on peut concevoir toute l'architecture du circuit en faisant une validation logicielle (simulation logique) puis dans une avant dernière étape en déduisant le dessin des divers masques de fabrication.

1.2.2.3 Les prés caractérisés (*Standard Cell*):

Ces circuits permettent de faire appel à des bibliothèques de fonctions standards que le concepteur d'ASIC place sur la puce en relation avec son application (Figure 1.4). Le traitement d'images utilise un grand nombre d'opérations arithmétiques du type addition/multiplication et dans ce cadre, des bibliothèques standards de circuits de calculs arithmétiques ont été développées et optimisées par les fabricants de circuits intégrés **[GHOZZI-03]**.

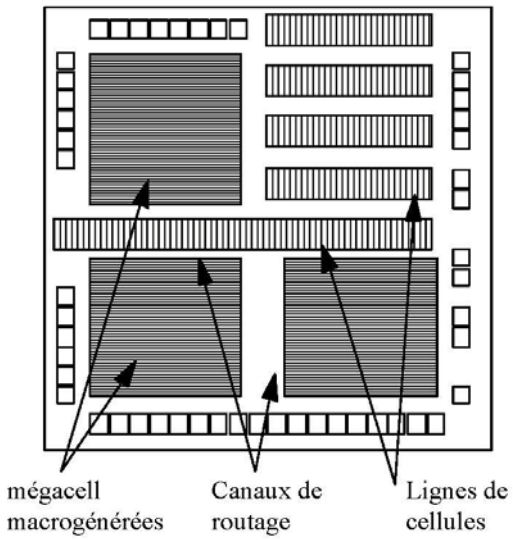


Figure 1.4 : Le circuit prés caractérisé.

1.2.3 les circuits programmables :

Un circuit logique programmable se définit comme un composant discret contenant des modules de logique combinatoires et séquentiels dont les interconnexions sont réalisées par programmation. Un dispositif à mémoire assure le contrôle et l´établissement des interconnexions entre les différentes fonctions logiques.

La complexité des PLD (*Programmable Logic Device*) n'a cessé d'évoluer, depuis leur apparition en 1970, jusqu'à aujourd'hui. Ils comptaient une centaine de portes (*gates*) « NAND à 2 entrées » à leur création et ne contenaient pas de bascules (*flip-flops*), pour arriver, actuellement, à un total de plus de 200 000 portes avec plusieurs milliers de bascules.

Les circuits programmables sont classés comme suit :

- les circuits programmables simples SPLD.
- les circuits programmables complexes CPLD.
- Les réseaux logiques programmables FPGA.

1.2.3.1 Les SPLD (*Simple Programmable Logic Devices*) :

Ce sont les composants à faible intégration et par conséquent les moins coûteux, la plupart des SPLD suivent la structure suivante :

- ✓ Un ensemble d'opérateurs « ET » sur lesquels viennent se connecter les variables d'entrée et leurs compléments.
- ✓ Un ensemble d'opérateurs « OU » sur lesquels les sorties des opérateurs « ET » sont connectées.
- ✓ Une éventuelle structure de sortie (Portes inverseuses, logique 3 états, registres...).

Donc, on peut classer les SPLD comme suit :

➢ **Les PAL (*Programmable Arrays Logic*) :**

La société MMI (*Monolithic Memories* devenu AMD (*Advanced Micro Devices*)) invente en 1970 les premiers circuits programmables PAL, ils sont basés principalement sur des architectures à plans logiques ET-OU qui permettent une implémentation naturelle et efficace seulement des équations logiques combinatoires sous la forme de somme de produits **[DUTRIEUX-97]**.

Les PAL disposent d'une matrice « ET » programmable et d'une matrice « OU » fixe (non programmable), ils sont caractérisés par un nombre fixe de termes produits et sommes (Figure 1.5), donc leur structure est simple et leur temps de propagation est prévisible.

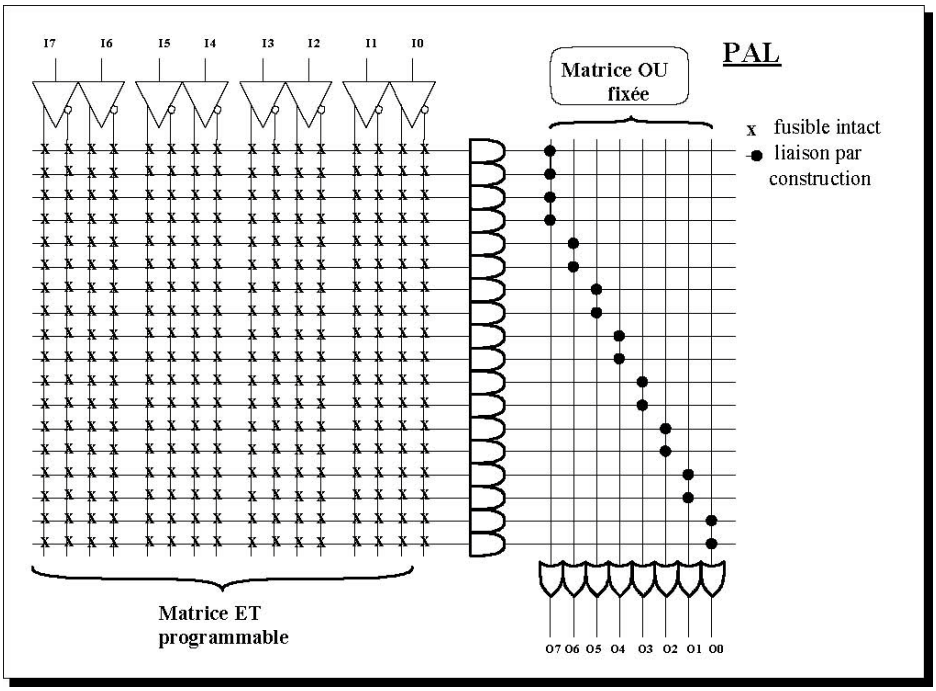


Figure 1.5 : Le concept d'un PAL.

La programmation d'une connexion repose sur un fusible, initialement conducteur et il devient un circuit ouvert par le passage d'un courant de valeur calibrée, ce qui rend les circuits PAL programmables qu'une seule fois.

- **Les GAL (*Generic Arrays Logic*) :**

L'inconvénient majeur des PAL est qu'ils ne sont programmables qu'une seule fois. Ce qui a ramené la firme LATTICE a pensé de remplacer les fusibles irréversibles des PAL par des transistors *MOS FET* qui peuvent être régénérés. Ceci a donné naissance aux GAL « Réseau Logique Générique ». Ces circuits peuvent donc être reprogrammés à volonté **[DARCHE-04]**.

On peut aussi noter que dans leur structure interne, les GAL sont constitués de transistor *CMOS* alors que les PAL classiques sont constitués de transistors bipolaires, donc ces derniers prend beaucoup de place sur la surface du Silicium que les transistors *CMOS*, par conséquent la consommation des GAL est beaucoup plus faible que les circuits PAL.

Par soucis de remplacer les PAL, LATTICE a équipé la plupart de ses GAL de macro-cellules de sortie programmable permettant d'émuler n'importe quel PAL **[DARCHE-04]**. Cette macro-cellule que l'on nomme OLMC (*Output Logic Macro Cell*) se compose d'une bascule D, d'un multiplexeur, et de différentes portes logiques (Figure 1.6).

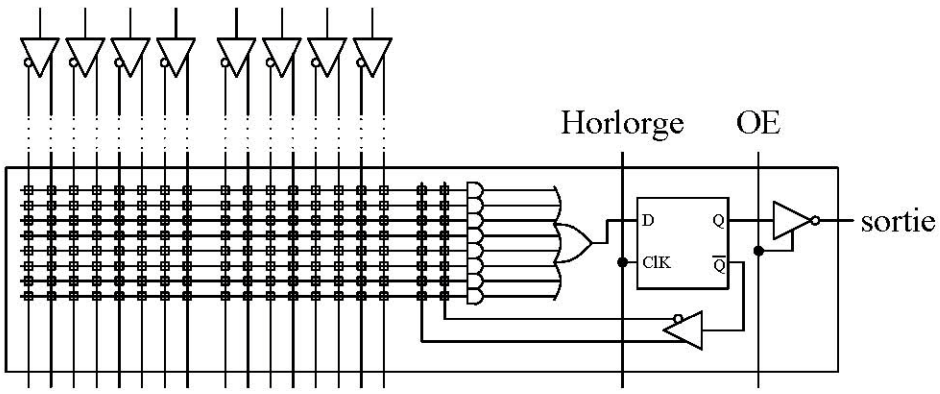


Figure 1.6 : Macro-cellule Programmable.

Un GAL peut réaliser des fonctions combinatoires ou séquentielles, actives hautes ou basses. Cette possibilité est de plus disponible indépendamment pour chaque sortie. Ces circuits programmables permettent enfin une utilisation souple et confortable.

Les GAL ont un bon critère de choix, la protection contre la duplication car ils sont dotés d'un bit de sécurité qui peut être activé lors de la programmation empêchant toute lecture du contenu du circuit. Ce bit est remis à zéro seulement en effaçant complètement le GAL.

Il est aussi constitué d'un ensemble de huit octets, appelé signature électronique, pouvant contenir des informations diverses sur le produit **[HACK-94]**.

- **Les PLA (*Programmable Logic Arrays*) :**

Les PLA sont des circuits programmables avec les deux matrices ET/OU programmables, donc le concepteur peut intervenir à deux niveaux, celui des termes sommes et celui des termes Produits (Figure 1.7).

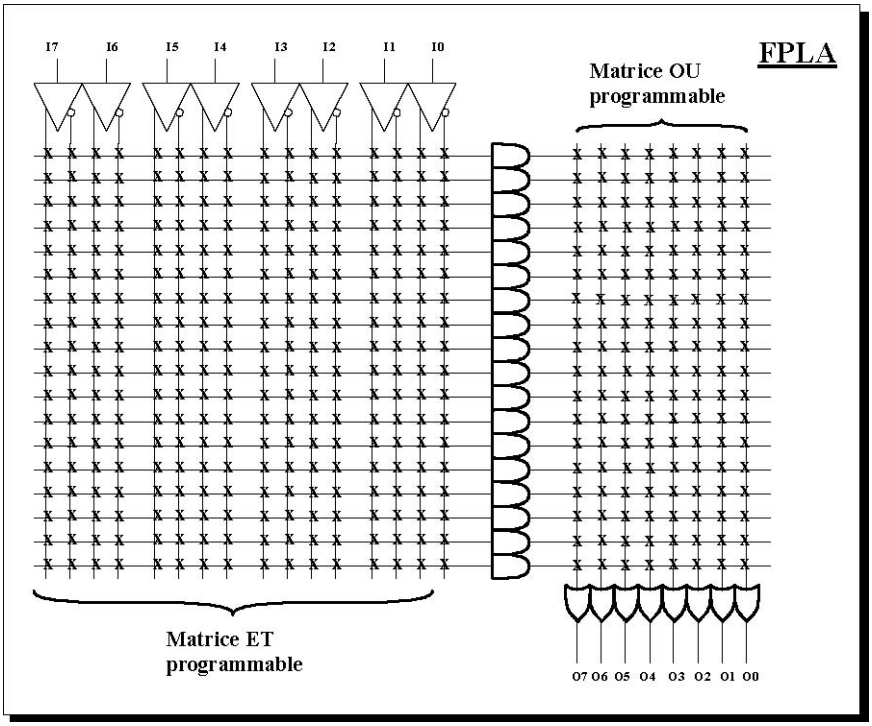


Figure 1.7 : Structure d'un PLA.

Le tableau 1.1 résume les différences entre les composants logiques programmables simples, qui résident essentiellement dans la programmabilité ou non de leurs plans ET, OU.

Type	**Plan ET**	**Plan OU**	**Technologie**	**Utilisation classique**
PAL	programmable	fixe	bipolaire	Décodage, machine à état
GAL	reprogrammable	fixe	CMOS	Décodage, machine à état
PLA	programmable	programmable	CMOS	Fonctions logiques complexes

Tableau 1.1 : Comparaison entre les SPLD.

1.2.3.2 Les CPLD (*Complex Programmable Logic Device*) :

Les CPLD peuvent être vu comme une intégration de plusieurs PLD simples dans une structure à deux dimensions, ils sont composés de blocs logiques répartis autour d'une matrice d'interconnexion PI (*Programmable Interconnect*), et chaque bloc logique LAB (*Logic Array Block*) est composé d'un ensemble de macro-cellule de type SPLD (Figure 1.8); En rappelant

que la macro-cellule, ainsi que la matrice d'interconnexion sont tous les deux programmables. Donc on peut dire que les CPLD sont similaires aux SPLD exception faite de leur niveau d'intégration plus élevé **[DARCHE-04]**.

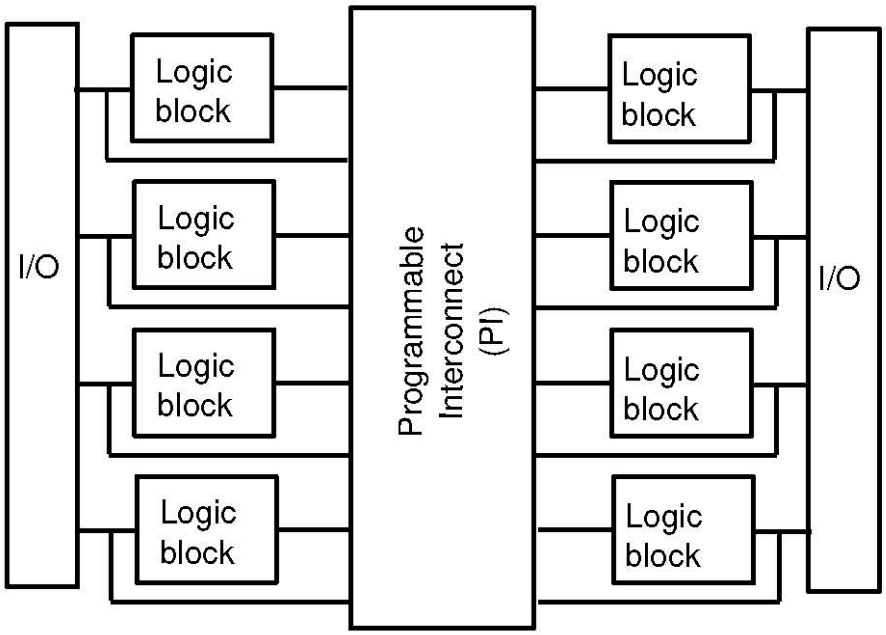


Figure 1.8 : Architecture d'un CPLD.

1.2.3.3 Les FPGA (*Field Programmable Gate Arrays*) :

Au milieu des années 80 les circuits de type FPGA réalisent la croisée des raisonnements génériques (PAL, CPLD) et spécifiques (ASIC) et profitent des avantages des deux technologies.

Les FPGA sont des circuits constitués de cellules interconnectées. La fonction de ces cellules ainsi que les interconnexions sont programmables ce qui donne une plus grande souplesse d'utilisation que les PLD.

C'est en 1984 que la société XILINX a commercialisé pour la première fois un circuit FPGA sous la dénomination de LCA (*Logic Cell Array*) (Figure 1.9). Bien que cette société ne soit plus la seule sur ce marché elle a continué à élargir sa gamme et propose actuellement une grande variété de produits qui utilisent la technologie de la mémoire vive (cf. § 1.3.4) **[DOUILLARD]**.

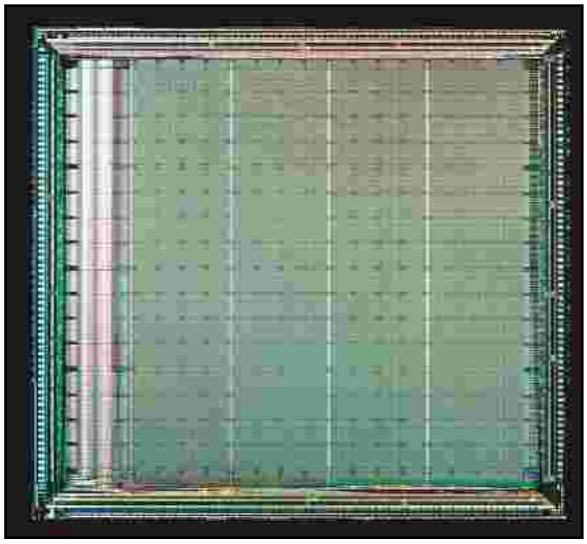


Figure 1.9 : FPGA XILINX.

Le choix des composants commerciaux étudiés s'est fait en fonction de leurs larges diffusions. D'après la figure 1.10, XILINX et ALTERA occupent les premières places en termes de parts de marchés. De plus, les architectures des FPGA de chez XILINX (cf. § 1.4) et ALTERA sont fondamentalement différentes.

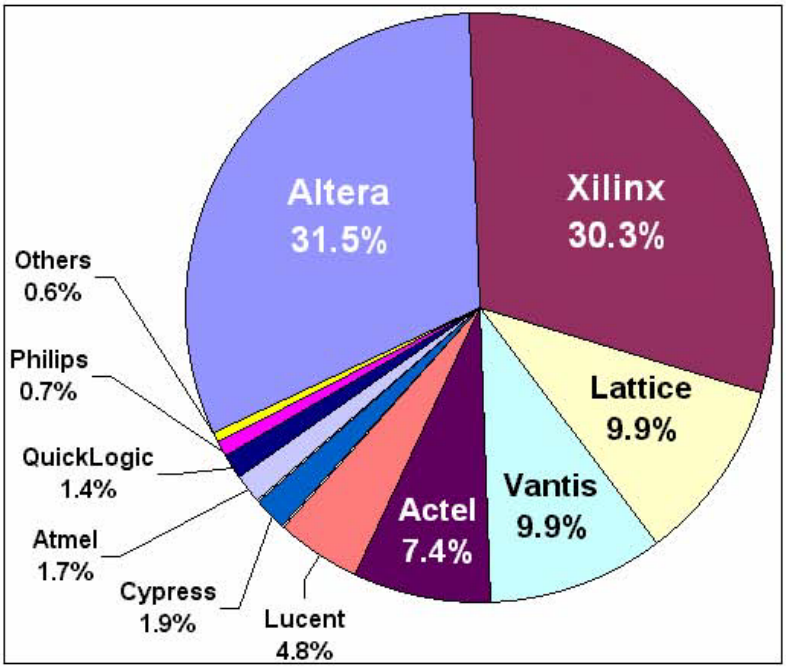


Figure 1.10 : Part de marché des fabricants de FPGA.

1.3 Les technologies d'interconnexions :

Au début des années 70, l'apparition de la connexion logique programmable a donné naissance à une industrie nouvelle qui n'a pas cessé de croître : c'est l'industrie des différents circuits logiques programmables car le premier critère de choix de ces derniers est la technologie utilisée pour matérialiser les interconnexions **[DUTRIEUX-97]**.

Les technologies d'interconnexions déterminent les aspects électriques de la programmation :

- ✓ La possibilité de modifier la fonction programmée.
- ✓ Le maintien de la fonction programmée en cas de coupure d'alimentation.
- ✓ La nécessité d'utiliser un programmateur spécial.

On distingue plusieurs technologies :

1.3.1 Les fusibles :

La connexion par fusibles a été la première méthode employée, On la trouve dans quelques circuits de faible densité, de conception ancienne car elle est en voie de disparition (Figure 1.11).

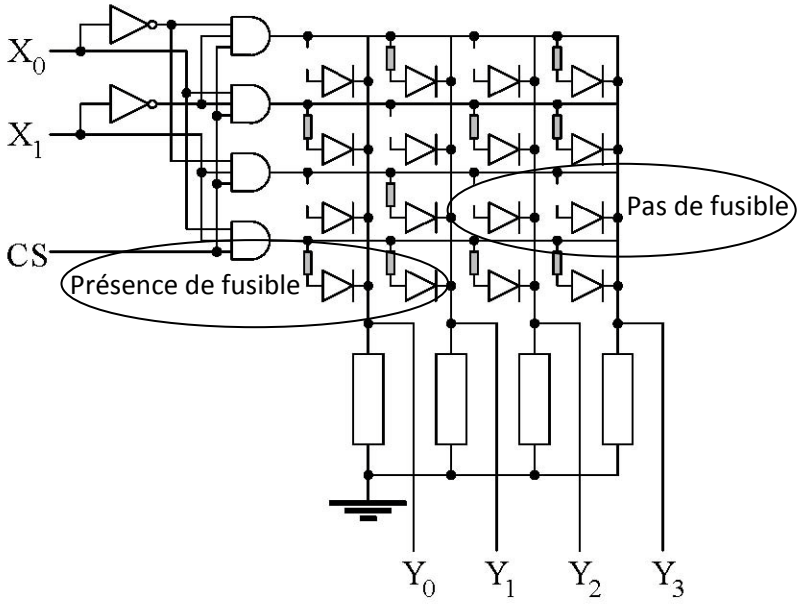


Figure 1.11 : PLD élémentaire à fusibles.

Toutes les connexions sont établies à la fabrication, le fusible étant disposé entre deux réseaux et en série avec la diode. Son principe est d'appliquer une tension de 12 à 25v (tension de claquage) aux bornes du fusible. Le claquage de ce dernier permet la suppression de la connexion. Donc l'inconvénient de cette technique est l'irréversibilité de la programmation.

1.3.2 Anti fusibles :

Le procédé des cellules anti-fusibles a été élaboré par la société ACTEL en 1986 sous l'appellation PLICE^{TM} (*Programmable Low Impedance Circuit Element*) **[DUTRIEUX-97]**. La technologie repose sur un circuit ouvert (en opposition aux cellules à fusible) réalisé avec un isolant ou diélectrique (Figure 1.12), et la programmation va donc permettre d'établir la connexion.

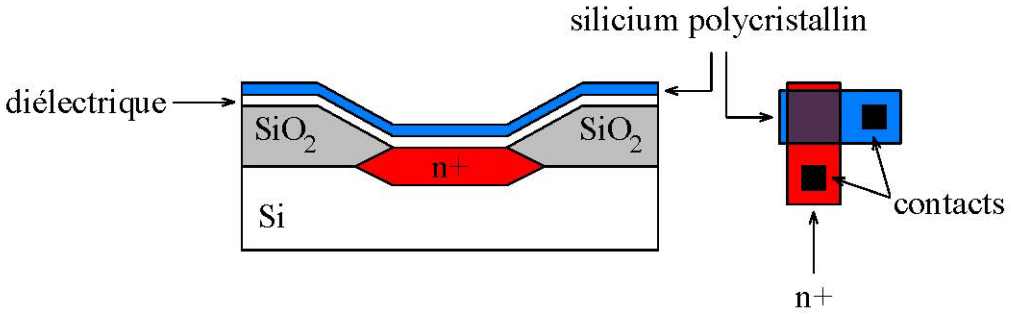


Figure 1.12 : La technique des anti-fusibles.

La technique de l'anti-fusible consiste à isoler deux lignes de connexions par une fine couche d'oxyde (isolant). Il n'y a donc pas de liaison électrique entre ces lignes de connexions. Par contre si une impulsion de haute tension (une vingtaine de volts) est appliquée à cet anti-fusible, la couche d'oxyde est trouée et les deux lignes sont reliées. La programmation est donc irréversible ce qui peut être un inconvénient.

1.3.3 MOS à grille flottante :

Dans toutes les techniques précédentes la programmation est irréversible, il est impossible de revenir en arrière. Toute modification suppose la programmation d'un nouveau circuit. Une autre technique offre la possibilité de multiprogrammation.

Pour cela, on utilise des transistors à effet de champ à structure MOS (*Metal oxyd semiconductor*). Les transistors MOS sont des interrupteurs, commandés par une charge électrique stockée sur leur électrode de grille. Si, en fonctionnement normal, cette grille est isolée, elle conserve sa charge éventuelle éternellement. Il reste au fondeur à trouver un moyen de modifier cette charge, pour programmer l'état du transistor **[WEBER-00]**.

Les transistors MOS à grille flottante disposent de deux grilles (Figure 1.13), dont l'une est isolée (grille d'un transistor MOS classique) et l'autre flottante (la grille de programmation). La programmation consiste à piéger les électrons dans la grille flottante en appliquant une tension de 13v entre cette dernière et la source, et une faible tension (≈ 5v) entre le drain et la source ce qui rend le canal conducteur ; le fort champ électrique entre la grille et le canal dévie les électrons présents dans ce dernier, et traversent la couche d'isolant de faible épaisseur (≈ 200A°) en se piégeant dans la grille flottante **[DUTRIEUX-97]**.

- La grille flottante ne contient aucun électron => le transistor n'a pas été programmé.
- La grille flottante contient des électrons => le transistor a été programmé.

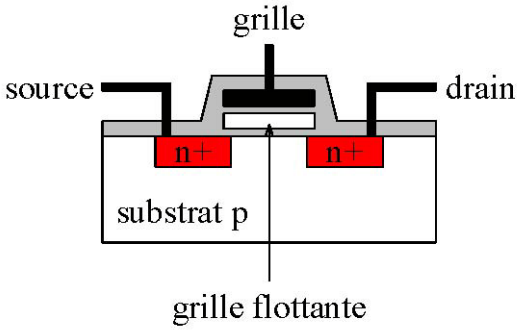


Figure 1.13 : MOS à grille flottante.

1.3.4 Mémoires statiques :

Dans les technologies à mémoire statique, l'état de chaque interrupteur est commandé par une cellule mémoire classique qui comporte cinq transistors : deux couples de transistors constituent chacun un inverseur logique et l'ensemble de ces deux inverseurs bouclés entre eux réalise un bistable statique, plus un transistor de programmation, son schéma de principe est illustré par la figure 1.14.

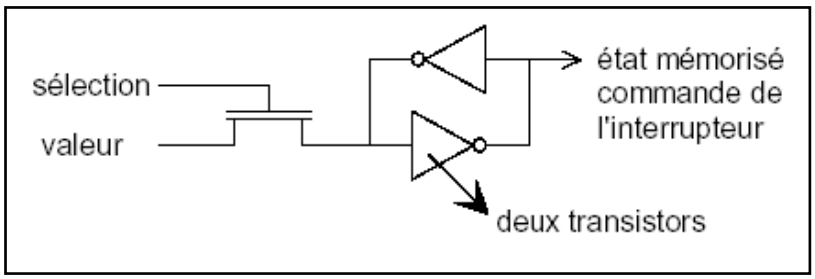


Figure 1.14 : Cellule SRAM.

La société XILINX est l'inventeur des connexions à SRAM **[DARCHE-04]**. L'intérêt de ces cellules qu'elles sont programmables à volonté, mais l'information est volatile. Ces circuits permettent des reconfigurations, partielles ou totales, en nombre illimité. Il est même envisageable de créer des fonctions dont certains paramètres sont modifiables en cours de fonctionnement, des filtres adaptatifs par exemple.

Les circuits à anti-fusibles partagent, avec ceux à SRAM, le sommet de la gamme des circuits programmables en vitesse et en densité d'intégration. Il est clair que ces circuits ne sont programmables qu'une fois.

1.4 Les réseaux logiques reconfigurables FPGA :

Le FPGA peut intégrer dans un seul circuit plusieurs fonctions logiques programmables par l'utilisateur. Sa mise en œuvre se fait très facilement à l'aide d'un programmateur, d'un micro-ordinateur et d'un logiciel adapté. La programmation consiste à connecter des équipotentielles afin d'obtenir les connexions et les fonctions souhaitées par l'utilisateur. Il existe actuellement deux technologies principales pour réaliser cette programmation. La

première utilise de la mémoire vive statique associée à des transistors utilisés en interrupteurs, la seconde met en œuvre des "anti-fusibles".

1.4.1 Les types d'architecture des FPGA :

Les FPGA sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique, car ils sont constitués d'un ensemble d'unités fonctionnelles qui, une fois programmées, permet de faire des opérations au niveau des bits.

La disposition des unités fonctionnelles dans le FPGA permet d'avoir différentes architectures de cette dernière, Classiquement on trouve trois topologies différentes :

1.4.1.1 Architecture de type îlots de calcul :

Dans cette architecture, le FPGA est constitué d'une matrice plane d'éléments. Ces différents éléments (que l'on détaillera par la suite) constituent les ressources logiques et de routages programmables du FPGA.
Ce type d'architecture a été choisi dès le départ par XILINX **[Xil.com]**.

1.4.1.2 Architecture de type hiérarchique :

Dans ce type d'architecture, il existe plusieurs plans dans le FPGA qui ne sont pas physiques, ils correspondent aux niveaux de hiérarchie logique. C'est à dire qu'un élément d'un niveau logique peut contenir des éléments de niveau logique inférieur, et Chaque niveau logique reprend la topologie d'une architecture du type îlots de calcul avec un routage dédié pour chaque niveau.
Cette architecture a été proposée par ALTERA **[Alt.com]**.

1.4.1.3 Architecture de type mer de portes :

Ce type de topologie a été utilisé par XILINX pour sa série 6000 **[GHOZZI-03]**; il est composé hiérarchiquement et le routage est de type logarithmique. Mais ces composants n'ont pas eu de succès, commercialement parlant, par manque d'outils de CAO capables de les exploiter correctement.

1.4.2 Les différents éléments des FPGA :

Dans le but de proposer une modélisation de FPGA il paraît évident de connaître leur constitution (éléments de base) et leurs modes de configuration. Il faut nécessairement étudier les composants d'un FPGA de type SRAM de XILINX, car le projet était basé sur ce type de circuit. L'architecture, retenue par XILINX (Figure 1.15), se présente sous forme de deux couches :

- ✓ une couche appelée circuit configurable,
- ✓ une couche réseau mémoire SRAM.

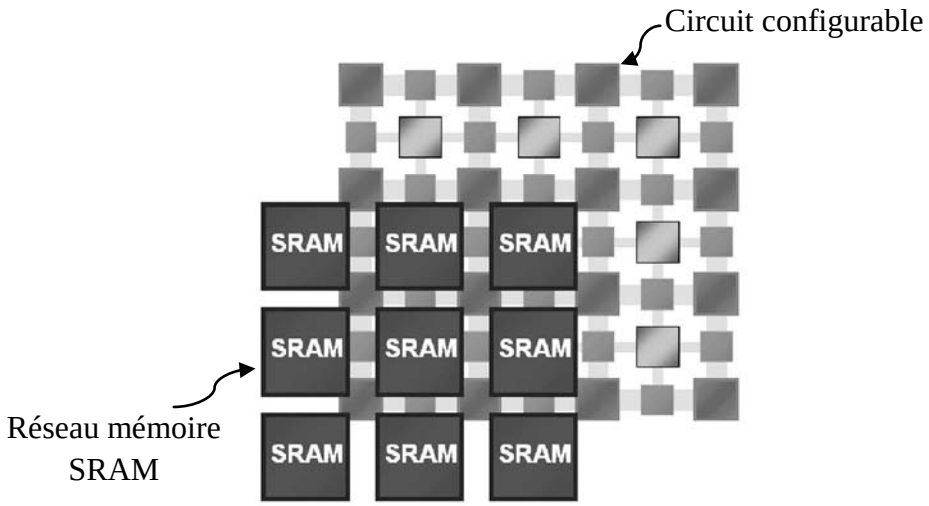


Figure 1.15 : Le concept d'un FPGA de type SRAM (XILINX).

1.4.2.1 Le circuit configurable :

Le circuit configurable (Figure 1.16) est composé de trois types d'éléments programmables :

- ✓ Les blocs logiques CLB (*Configurable Logic Blocks*) qui permettent la réalisation des fonctions combinatoires et séquentielles,
- ✓ Les blocs d'entrée/sortie IOB (*Input/Output Blocks*) qui assurent l'interface avec l'extérieur du circuit,
- ✓ Les matrices d'interconnexions (*Programmable Interconnect*) qui permettent d'interconnecter les blocs.

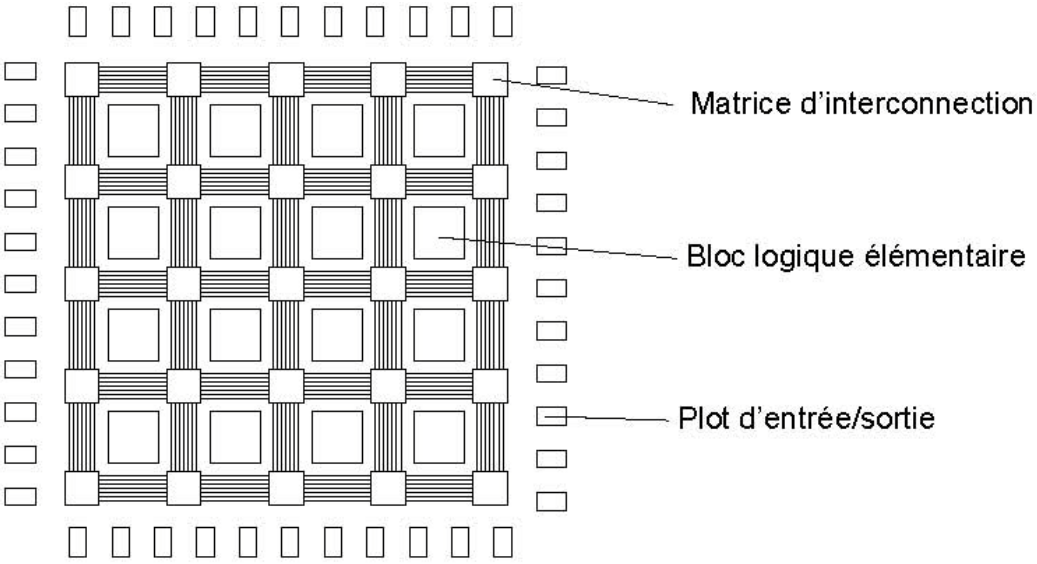


Figure 1.16 : Structure interne du Circuit configurable.

- **Les blocs logiques CLB (C*onfigurable Logic Blocs*):**

C'est l'unité fondamentale du bloc logique qui fournit des éléments utilitaires pour la logique combinatoire et la logique séquentielle, y compris les éléments du stockage de base. Chaque CLB est composé de quatre slices répartis sur deux colonnes, chaque slice étant composé à son tour de deux cellules logiques (*Logic Cell*) disposées en colonne (Figure1.17). Cette organisation en colonnes permet d'implémenter efficacement des additionneurs grâce à des chaînes de propagation de retenue rapides.

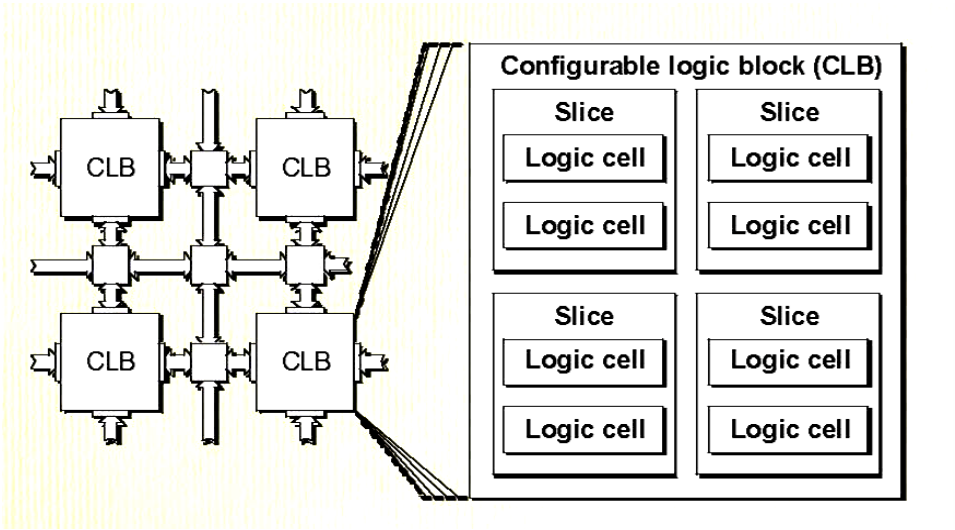


Figure 1.17 : Un bloc CLB (C*onfigurable Logic Blocs*).

Les cellules logiques d'un FPGA sont généralement constituées d'une partie calculatoire et d'une partie de mémorisation (Figure 1.18). La partie calculatoire est assurée par des générateurs de fonctions appelés LUT (*Look Up Table*) qui contiennent, après configuration, la table de vérité de la fonction logique qu'elles doivent réaliser, la LUT est donc une petite mémoire de 2^n mots de 1 bit, adressée par n bits provenant de la matrice de routage. Quant à la partie de mémorisation, il s'agit de bascules D (*flip-flop*) contrôlées par un signal horloge, Ces cellules disposent par ailleurs de chaînes de retenue (*Carry*) **[DETREY-07]**.

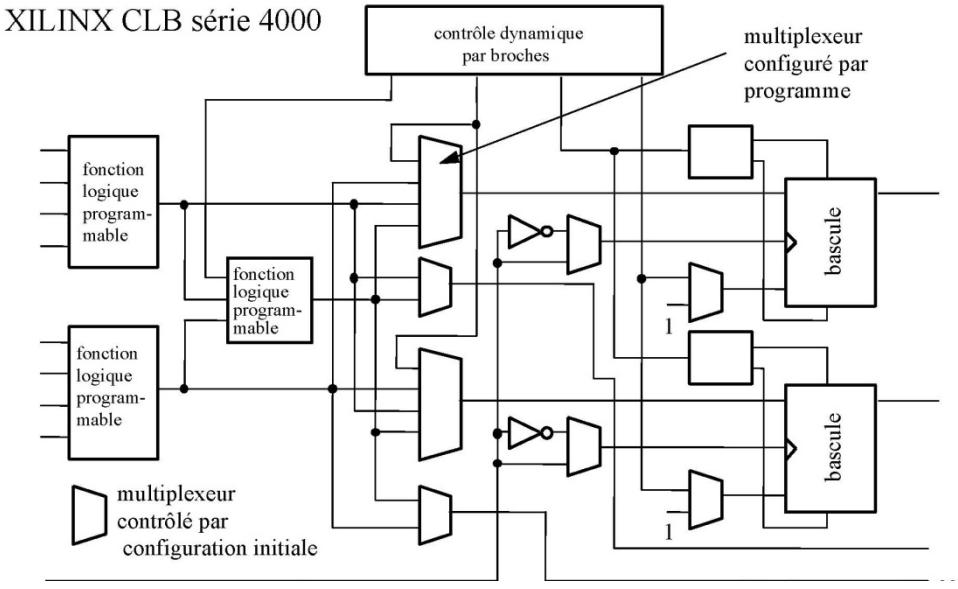


Figure 1.18 : Schéma d'une cellule logique (XC4000 de XILINX).

- **Les blocs d'entrée/sortie IOB (*Input/Output Blocks*) :**

Les blocs d'entrée/sortie sont positionnés à la périphérie du circuit FPGA. Ces IOB permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant (Figure 1.19), et Chaque IOB peut-être configuré en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haute impédance).

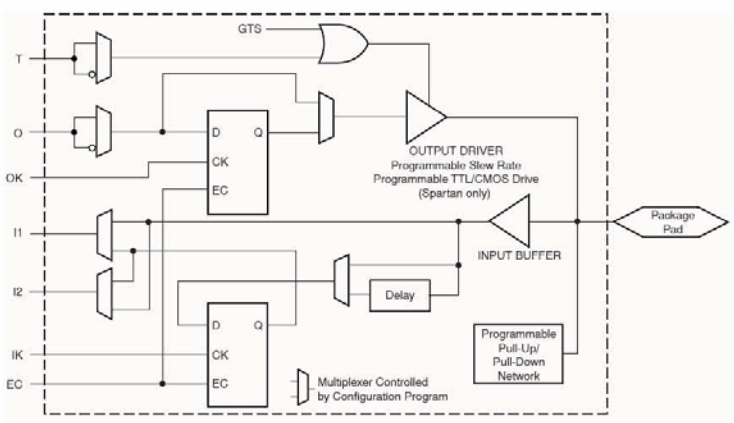


Figure 1.19 : Schéma d'un bloc d'entrée/sortie (SPARTAN).

- Configuration en entrée :

Premièrement, le signal d'entrée traverse un buffer qui, selon sa programmation, peut détecter soit des seuils TTL soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée. Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (*Program Controlled Multiplexer*). Un bit positionné dans une case mémoire commande ce dernier.

- Configuration en sortie :

Nous distinguons les possibilités suivantes :

- ✓ Inversion ou non du signal avant son application à l'IOB ;
- ✓ Synchronisation du signal sur des fronts montants ou descendants d'horloge ;
- ✓ Mise en place d'un "pull-up" ou "pull-down" dans le but de limiter la consommation des entrées/sorties inutilisées ;
- ✓ Chaque sortie peut délivrer un courant de 12mA. Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

- **Les interconnexions programmables (*Programmable Interconnect*):**

Dans les FPGA, on trouve des connexions internes, ils sont composés de segments métallisés et des matrices programmables qui sont réparties sur la totalité du circuit, positionnées horizontalement et verticalement entre les divers CLB. Ces matrices programmables

permettent les connexions entre les diverses lignes, qui sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive (RAM).

Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible. Pour parvenir à cet objectif, XILINX propose trois sortes d'interconnexions selon la longueur et la destination des liaisons **[TARIGHT-99]**. Nous disposons :

- ✓ d'interconnexions à usage général,
- ✓ d'interconnexions directes,
- ✓ de longues lignes.

- **Les interconnexions à usage général :**

Cette configuration fonctionne en une grille de cinq segments métalliques verticaux et quatre segments horizontaux positionnés entre les rangées et les colonnes de CLB et de l'IOB (Figure 1.20).

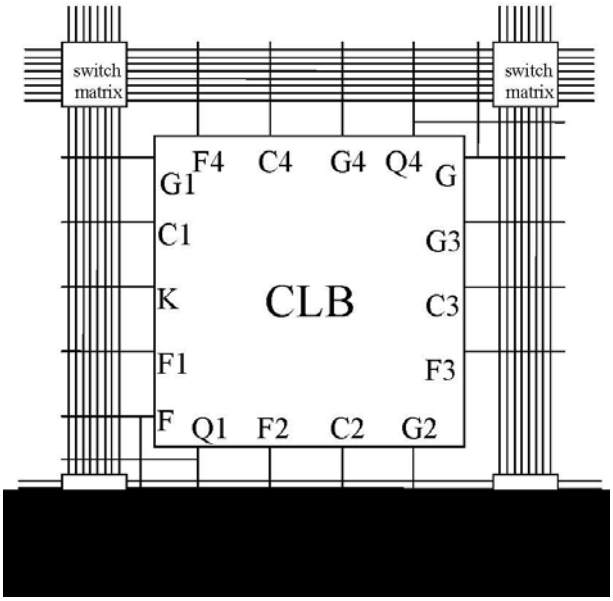


Figure 1.20 : Connexion à usage générale.

A chaque intersection on trouve des matrices de commutation (*Switch Matrix*) qui permettent de raccorder les segments entre eux selon diverses configurations (Figure 1.21); on trouve aussi des buffers implantés en haut et à droite de chaque matrice de commutation dans le but d'éviter que les signaux traversant les grandes lignes ne soient affaiblis. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre.

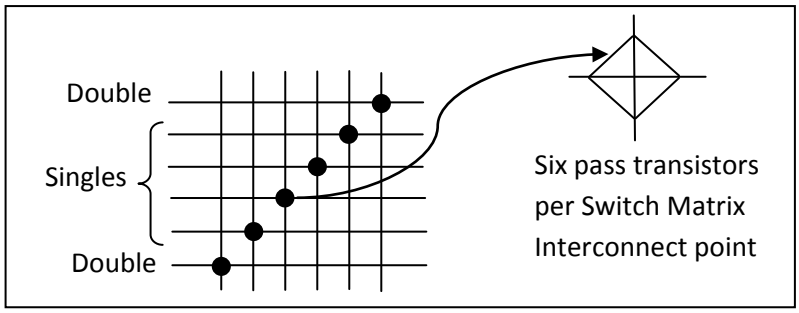


Figure 1.21 : Concept d'une matrice de commutation.

- **Les interconnexions directes :**

Ces interconnexions sont des lignes qui permettent des liaisons directes entre les CLB et les IOB, ou entre les différents blocs logiques (Figure 1.22).

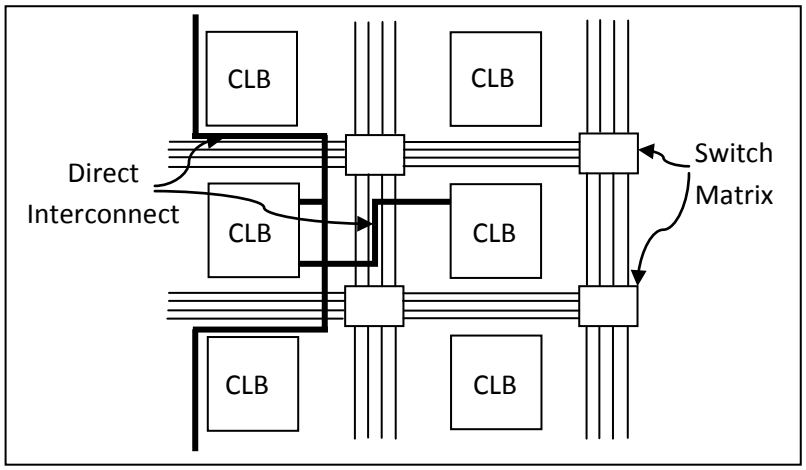


Figure 1.22 : Les interconnexions directes.

- **Les longues lignes :**

Ces interconnexions sont de longs segments métallisés qui permettent la transmission des signaux entre les différents éléments avec un minimum de retard dans le but d'assurer un synchronisme aussi parfait que possible (Figure 1.23).

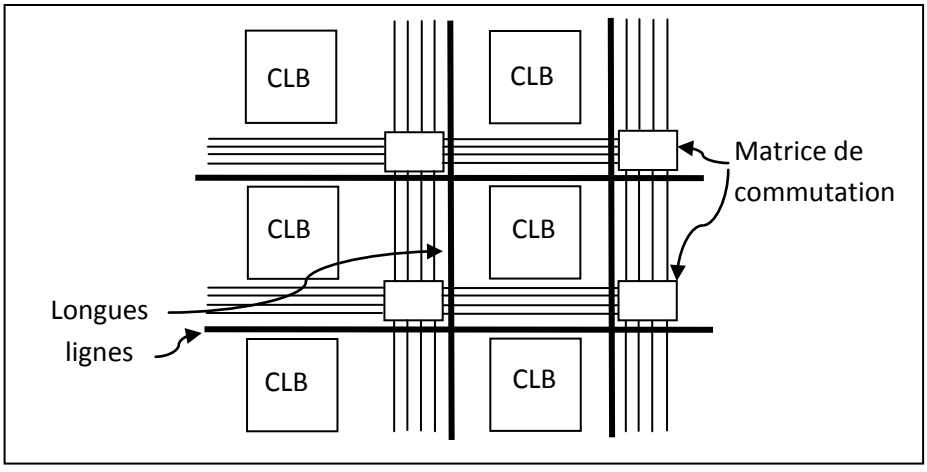


Figure 1.23 : Les longues lignes.

L'horloge est un élément essentiel pour le bon fonctionnement d'un système électronique. Les circuits FPGA sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routage spécialement adaptées au transport d'horloges sur de longues distances.

L'horloge est **un oscillateur à quartz,** placé dans un angle de la puce, il peut être activé lors de la phase de programmation pour réaliser un oscillateur. Il utilise deux IOB voisins, pour réaliser l'oscillateur. Cet oscillateur ne peut être réalisé que dans un angle de la puce où se trouve l'amplificateur prévu à cet effet. Il est évident que si l'oscillateur n'est pas utilisé, les deux IOB sont utilisables au même titre que les autres IOB.

1.4.2.2 Le réseau mémoire SRAM :

Les FPGA ont changé et ne sont plus seulement utilisés pour des applications de type "logique de glue" (*Glue Logic*), comme ce fût le cas à leur début, mais aussi pour des applications plus importantes qui demandent souvent des capacités de stockage (comme le traitement d'images). La nécessité d'intégrer des blocs de mémoires directement dans l'architecture des FPGA est vite devenue cruciale. De cette façon les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

La programmation du circuit FPGA, appelé aussi LCA (*Logic Cell Array*), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

La programmation d'un circuit FPGA est volatile, la configuration du circuit est donc mémorisée sur la couche réseau SRAM et stockée dans une ROM externe. Un dispositif interne permet à chaque mise sous tension de charger la SRAM interne à partir de la ROM.

1.4.3 Les FPGA actuelles :

Les modèles de FPGA récents intègrent également des multiplieurs¸ des DSP et même des noyaux de processeurs **[DETREY-07].**

1.4.3.1 Bloc de petits multiplieurs dans un FPGA :

Pour permettre l'implémentation efficace de circuits très calculatoires et massivement parallèles, comme on en trouve en traitement du signal ou en traitement d'images, les générations actuelles de FPGA embarquent toutes de petits multiplieurs entiers. Ces multiplieurs, accessibles par la matrice de routage, permettent donc d'accélérer les calculs, mais surtout de réduire grandement le nombre de cellules logiques utilisées par le circuit.

1.4.3.2 Blocs des DSP dans un FPGA :

Les générations les plus récentes de FPGA proposent même des blocs DSP (*Digital Signal Processing*) dédiés au traitement du signal. Ces blocs sont généralement composés d'un multiplieur et d'un accumulateur MAC (*Multiplier and ACcumulator*) et permettent ainsi de réaliser des filtres de manière très efficace **[MCALLISTER-06]**.

1.4.3.3 Blocs de cœurs de processeurs dans un FPGA :

Enfin, devant l'utilisation croissante des FPGA pour l'implémentation de systèmes embarqués comme les SOC (*System on Chip*), certains modèles de FPGA intègrent directement un ou plusieurs cœurs complets de processeurs RISC, soit d'une façon logicielle (*SoftCore*) ou d'une manière matérielle (*HardCore*), comme le MicroBlaze de XILINX ou le NIOS II d'ALTERA, véritables processeurs implémentés dans la matrice de cellules logiques **[POUSSIER-02]**.

1.5 Les critères de choix :

Le concepteur d'un système numérique doit faire un ou plusieurs choix parmi les circuits qui ont été présentés pour la réalisation de circuit logique; les critères le plus souvent utilisés sont présentés ci-après :

1.5.1 Coût de développement et fabrication :

C'est le coût des dépenses engagées pour concevoir le système et réaliser les outils nécessaires à sa fabrication et son test. Ce critère intervient dans le choix du circuit que le nombre de pièces à réaliser est faible, ce qui favorise l'utilisation des processeurs, les différents circuits programmables (SPLD, CPLD, FPGA) que les ASIC (surtout les *Full Custom*).

L'avantage principal des FPGA est un faible coût pour une production à faible unité, ou pour le développement de prototypes, car ce dernier est reprogrammable indéfiniment. Par contre, il devient d'un coût plus important qu'un circuit ASIC pour une production de plus grande série (Figure 1.24).

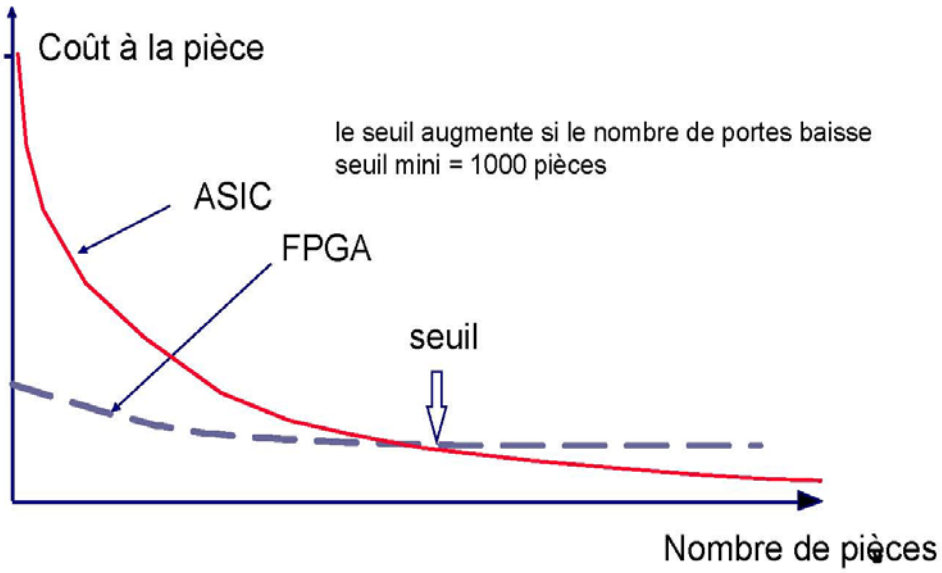


Figure 1.24 : Coût des FPGA par rapport aux ASIC

1.5.2 Temps de développement :

Si les circuits ASIC présentent des avantages évidents en termes de capacité d'intégration et de vitesse de calcul, ils présentent par contre des inconvénients d'une part en termes de temps de développement car la réalisation d'un circuit ASIC nécessite une phase de fabrication qui peut être de plusieurs semaines (Figure 1.25), et d'autre part en termes de coût de développement qui est relativement important.
Le temps de fabrication à l'aide d'un circuit FPGA se résume à sa programmation ce qui est négligeable.

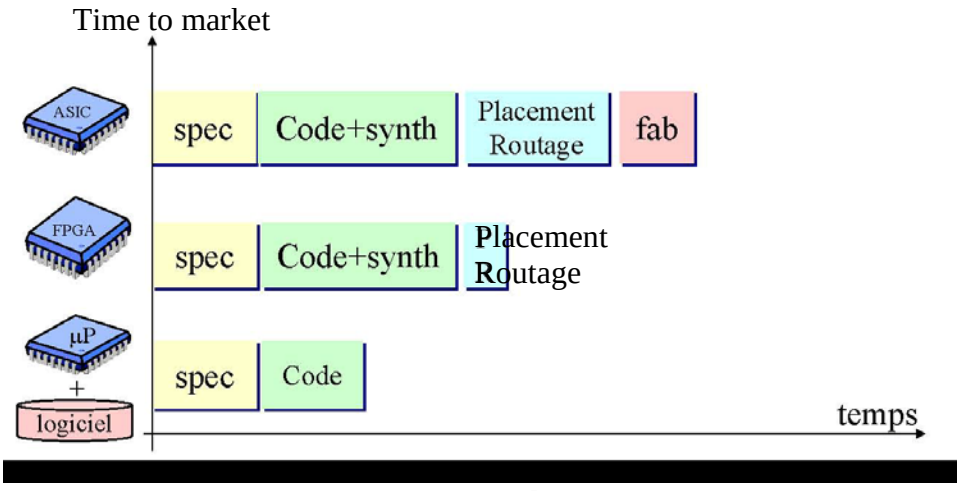


Figure 1.25 : Temps de conception.

1.5.3 Souplesse d'utilisation :

Favorise les circuits programmables (SPLD, CPLD, FPGA) dont on peut modifier plus facilement des fichiers que des circuits.

1.5.4 Taille :

Il y a une forte dépendance entre la taille du système et la densité d'intégration. L'augmentation de la densité d'intégration produit des systèmes de taille réduite. La figure suivante représente la densité d'intégration (Figure 1.26) dans les différentes familles des circuits logiques.
Les ASIC sont les mieux adaptés puisqu'ils sont totalement adaptés à l'application (plus particulièrement les *Full custom* et les *Standard Cell*).

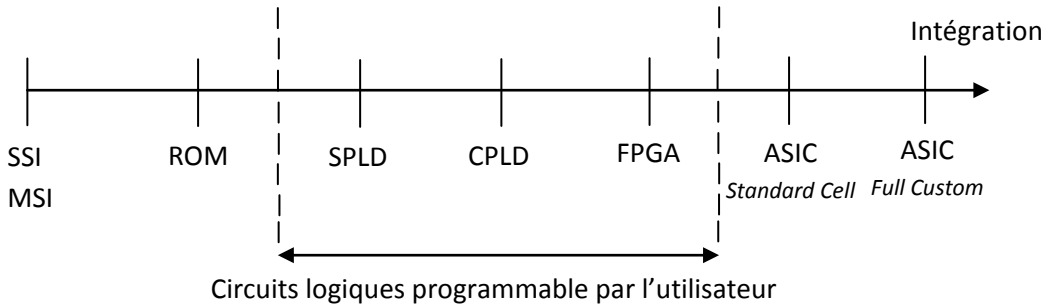


Figure 1.26 : L'intégration dans les circuits logiques.

1.5.5 Consommation :

Critère particulièrement sensible dans les applications possédant une alimentation autonome. Il conduit à favoriser des solutions ASIC pour les mêmes raisons que le critère précédent **[DOUILLARD]**.

1.5.6 Vitesse de fonctionnement :

Les CPLD sont des composants pour la plupart reprogrammables électriquement ou à fusibles, peu chers et très rapides (fréquence de fonctionnement élevée) mais avec une capacité fonctionnelle moindre que les FPGA.

1.5.7 Capacité mémoire :

Les FPGA à SRAM contiennent des mémoires pour stocker leur configuration. La plupart des familles récentes offrent à l'utilisateur la possibilité d'utiliser certaines de ces mémoires en tant que telles **[WEBER-00]**.

1.6 Conclusion :

Dans ce chapitre, on a pu voir les avancées actuelles dans la technologie des circuits intégrés, et en particulier des circuits logiques programmables qui avaient réduit fortement l'encombrement des cartes numériques pour une utilisation souple et confortable.

En contrepartie, cette évolution, qui ne permet plus une programmation simple, a demandé l'étude et l'utilisation de nouveaux langages de conception. Ces langages, de manière semblable à l'informatique, permettent une programmation à un niveau d'abstraction de plus en plus élevé. Ils seront détaillés dans le prochain chapitre.

Chapitre 2

Méthodologie De La Conception

2.1 Introduction :

Dans l'approche traditionnelle, un système numérique est un assemblage sur une carte de différents composants discrets représentant chacun une fonction particulière plus ou moins complexe telle qu'additionneur, mémoire, composant d'interface, gestionnaire d'interruption, processeur... Si une erreur de conception était faite, il était au minimum nécessaire d'ajouter des fils entre les composants, ou au pire, de refaire une carte pour régler le problème, c'est-à dire reprendre complètement son routage. Plus le système numérique est complexe, plus ces composants sont nombreux, plus la carte est chère, et plus les perturbations électromagnétiques sont importantes. Un besoin existait donc de pouvoir modifier la logique sans modifier les cartes et aussi de diminuer le nombre de composants sur une carte numérique. En effet, moins il y a de composants remplissant un même cahier des charges, moins la carte est chère, et plus les fonctions sont intégrées, plus il est possible de proposer une carte moins encombrante. Les améliorations des processus de fabrication des composants électroniques ont permis de répondre de mieux en mieux à ces besoins **[BEN ATITA-07]**.

Donc après avoir pris connaissance des différents types de circuits numériques et de leurs structures, nous présenterons dans ce chapitre les différentes méthodologies de conception d'un circuit. Nous exposons par la suite les outils de développement nécessaires à la conception ainsi que les différents langages utilisés.

2.2 Méthodes de conception :

Le travail est dans sa première étape ; il s'agit de définir et d'évaluer les différentes phases permettant la mise en place d'un flot de conception des circuits logiques programmables, pour cela on distingue deux méthodes de conception pour leurs développements :

2.2.1 La conception des circuits à faibles densités :

Le principe de programmation d'un SPLD est relativement simple car il faut déterminer les fusibles à claquer. Mais ce travail peut devenir long et fastidieux dans le cas d'utilisation de réseaux logiques programmables complexes ou de fonctions à réaliser complexes.

La programmation des SPLD nécessite un logiciel adapté pour le développement du projet et un programmateur permettant de programmer le circuit. En outre il est conseillé de suivre la démarche décrite par l'organigramme suivant (Figure 2.1) :

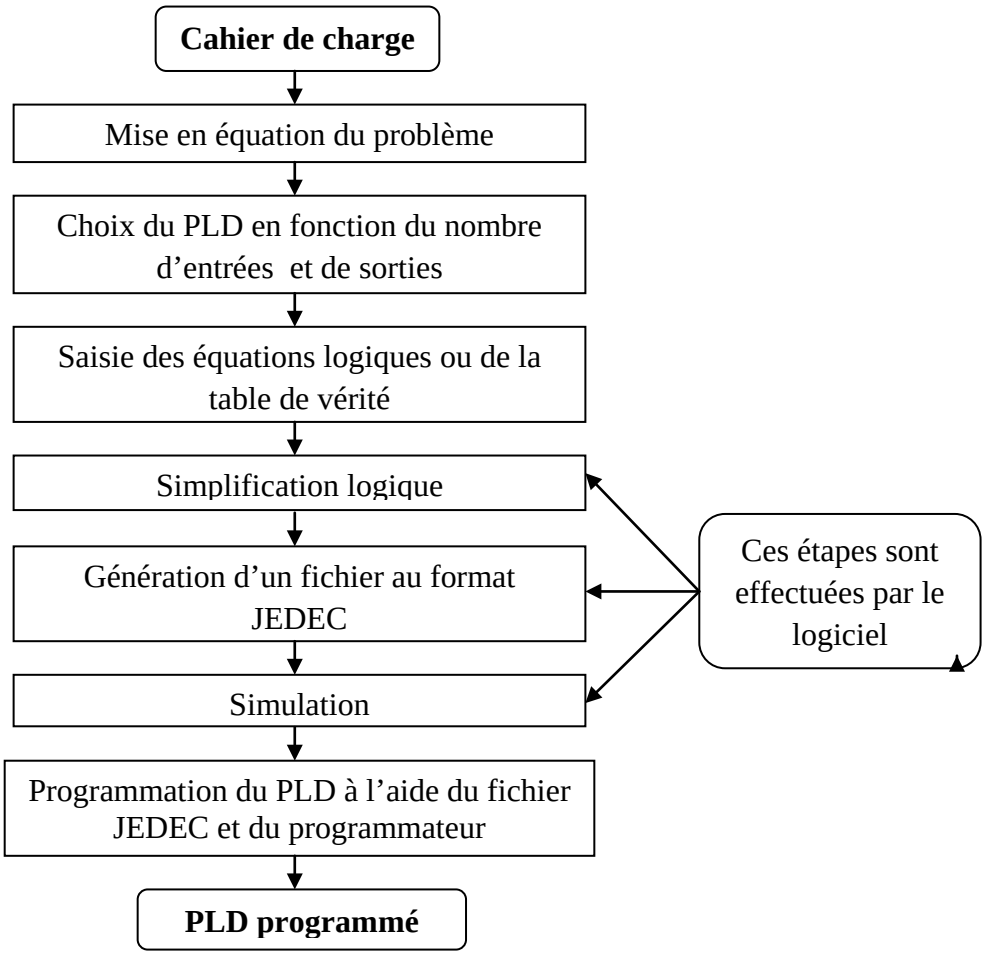


Figure 2.1 : L'organigramme de la conception des circuits à faibles densités.

Le fichier JEDEC est obtenu par un compilateur dont le point d'entrée est un fichier texte écrit en langage ABEL, le fichier standardisé au format JEDEC (Nom.JED) est un ensemble de données binaires indiquant au programmateur les fusibles à « griller ». Ce fichier est produit par un outil de développement (Comme par exemple WINCUPL qui est un logiciel de conception adapté à tous les SPLD et CPLD. C'est un logiciel propriétaire à ATMEL, complet et facile à utiliser, la description de conception se fait à l'aide du langage CUPL), à partir d'équations par exemple et permet aussi d'en simuler le fonctionnement.
Les programmateurs utilisés sont les mêmes que ceux permettant la programmation des EPROM.

2.2.2 La conception des circuits à hautes densités :

Le flot de conception classique des circuits de hautes densités (CPLD, FPGA, ASIC) est présenté par l'organigramme suivant (Figure 2.2):

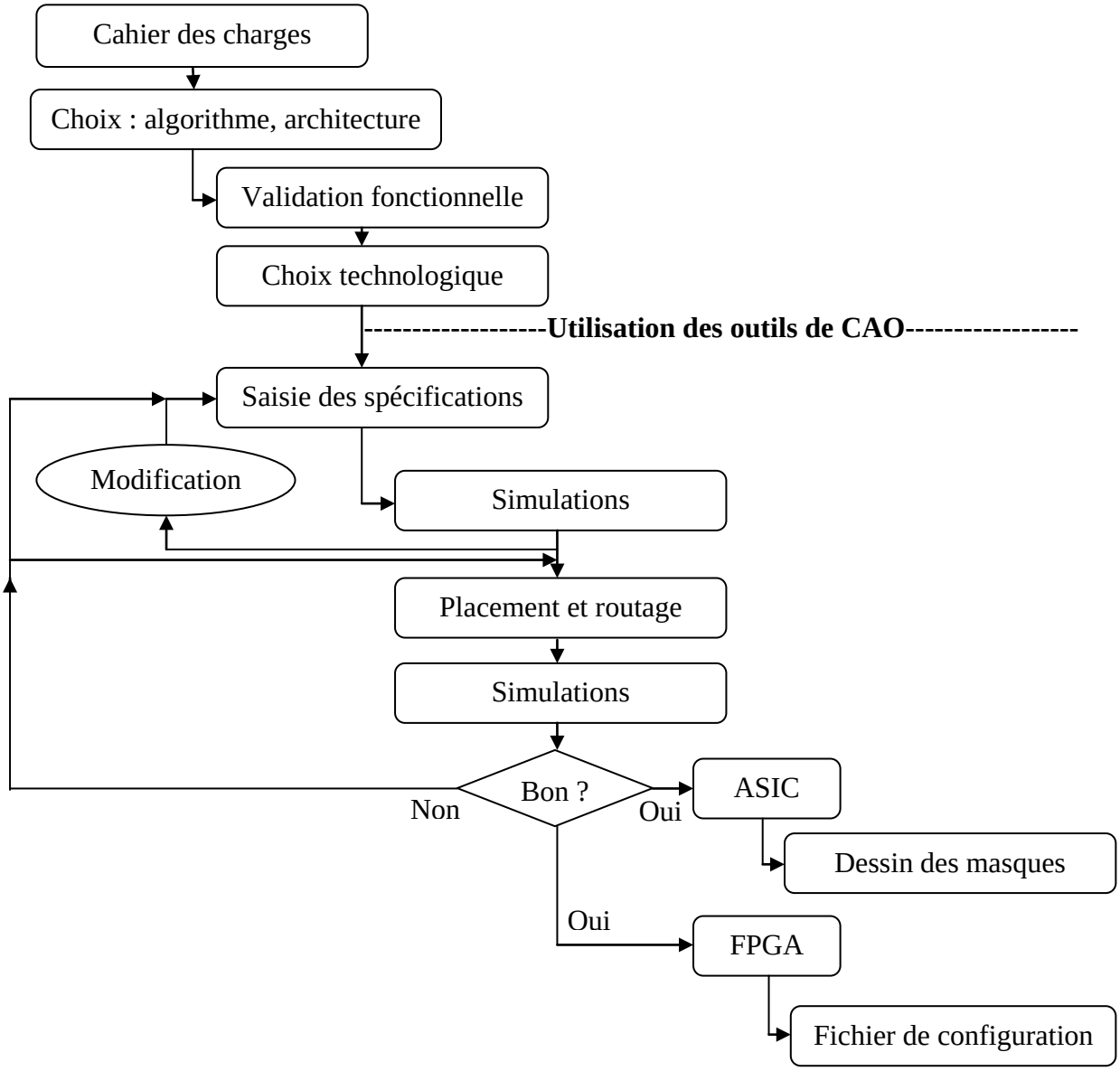


Figure 2.2 : L'organigramme de la conception des circuits à hautes densités.

2.2.2.1 Le cahier des charges :

Le point de départ est le cahier des charges, qui définit la fonctionnalité du circuit ainsi que les conditions de fonctionnement (telle que la température de fonctionnement, les vibrations et le rayonnement électromagnétique subis….), il doit spécifier aussi les contraintes d'encombrements, de tension d'alimentation, de vitesse de fonctionnement…

2.2.2.2 La validation fonctionnelle :

Le concepteur commence par écrire un modèle comportemental (ou fonctionnel) du circuit en choisissant un algorithme et une architecture convenable. Cette étape consiste à identifier les principales fonctions et faire un découpage en blocs du circuit; cette découpe est généralement

réalisée en plusieurs étapes amenant à des descriptions de plus en plus détaillées. La découpe est terminée lorsque chaque fonction élémentaire peut être réalisée à l'aide d'éléments disponibles dans la bibliothèque utilisée pour la synthèse logique (fonctions combinatoires élémentaires, opérateurs arithmétiques, compteurs, registres, automates, ...). Cette bibliothèque est fournie par le fondeur choisi pour la fabrication du circuit. Le but de cette étape est de valider la partie fonctionnelle du cahier des charges en respectant toujours les contraintes du temps et de surface.

2.2.2.3 Le choix technologique :

Ensuite, on va établir le choix technologique c'est-à-dire le choix de la manière de décrire un circuit. Soit on veut faire fabriquer ce circuit avec un ASIC (*Application Specific integrated circuit*), ou bien l'implémenter sur un FPGA (*Field Programmable Gate Arrays*).

2.2.2.4 La synthèse :

L'étape suivante est la synthèse, indépendante de la technologie du circuit cible, contient la saisie de l'application dans un outil de CAO (Conception Assistée par Ordinateur) (cf. § 2.3.1), en tenant compte de la nature de la fonction à réaliser et du composant qui va accueillir cette fonction, car c'est des critères importants pour faire un choix du mode de description qui facilite la conception de ce circuit. On peut définir trois descriptions :

Une description schématique utilise des composants de base (hard macro, composants optimisés) fournis pour un circuit donné ou du moins une famille de ce dernier. On utilisera ce type de description lorsque la nature de la fonction à implanter est très structurée. On peut dire que ce fichier source (schéma) est difficilement portable même si l'ensemble des composants de base utilisés dans les conceptions numériques se limitent à des fonctions restreintes et génériques (bascules, compteurs…), car chaque constructeur propose sa propre librairie de composants et les noms utilisés pour ces composants génériques ne sont en générale acceptés que par le compilateur du circuit cible.

Une description HDL utilise un langage de type comportemental permet de rendre le code source indépendant de la cible. Un code HDL normé (VHDL, VERILOG, …) sera accepté par la plupart des outils (simulateurs, synthétiseurs…).

Une description mixte consiste à définir certains éléments hiérarchiques en schématique et d'autres de manière syntaxiques. Il pourra par exemple s'avérer judicieux de définir schématiquement la structure globale d'une conception et de définir de manière comportementale chaque bloc fonctionnel **[DUTRIEUX-97]**.

La synthèse aboutit à une description du circuit au niveau logique, couramment appelée : "netlist". Cette étape prend en compte les contraintes imposées par le cahier des charges (surface, temps). La synthèse est validée toujours par une simulation logique qui utilise les caractéristiques temporelles fournies par la bibliothèque.

2.2.2.5 La première simulation :

Cette simulation peut être direct ou avec un banc d'essai (*test bench*). La simulation directe permet de charger le composant à simuler dans un simulateur (par exemple : MODELSIM) directement et faire varier entrée par entrée à chaque pas de simulation en définissant le temps qui dépendra de la fréquence du signal d'horloge.

La simulation par un fichier de test (*test bench*) consiste à créer un composant B sans entrées –sorties (Figure 2.3) en déclarant le composant A à simuler et les signaux internes (autant de signaux que les entrées-sorties du composant A). Ce banc d'essai est un module écrit en VHDL qui permet de faire varier les signaux internes connectés aux entrées afin de visualiser les résultats sur une fenêtre des courbes **[KARABERNOU-09]**.
Cette dernière méthode de simulation est recommandée car elle est méthodique et elle nous permet de vérifier le fonctionnement de la description à chaque étape de la procédure de développement.

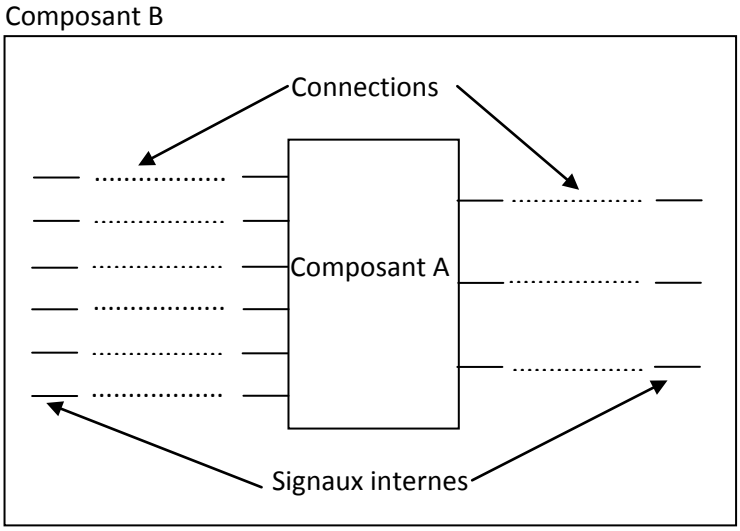


Figure 2.3 : Simulation par un banc de test.

2.2.2.6 Le placement et le routage :

Reste ensuite à déterminer l'emplacement physique des blocs logiques sur le silicium (placement) et à déterminer les chemins suivis par les interconnexions (routage). Les problèmes du placement et du routage ne sont pas indépendants : la qualité du placement pouvant fortement influencer sur le routage dans le circuit.

Le placement consiste aussi à affecter les signaux utilisés dans le circuit à ses plots d'entrées/sorties.

2.2.2.7 La deuxième simulation :

A l'issue de ces opérations (après simulation), il est possible d'extraire les retards apportés par les connexions, et de modifier la netlist en conséquence. Cette opération s'appelle la rétro-annotation. En effet, à la synthèse, ce sont des valeurs par défaut correspondant à des

capacités estimées moyennes qui sont utilisées. Une simulation post-routage utilisant les retards réels permet de vérifier si le circuit définitif répond bien aux contraintes du cahier des charges. Sinon, il faut refaire la synthèse en tenant compte des résultats de la rétro-annotation. Si les résultats de la rétro-annotation sont parfaits, on va établir l'étape suivante !

2.2.2.8 La fabrication ou la programmation du circuit :

Si c'est un ASIC, on réalise le dessin des masques en respectant les règles du dessin afin d'établir la production du circuit. Et si c'est un FPGA un fichier de configuration, contenant l'ensemble des informations relatives à l'implémentation, est produit. Celui-ci peut alors être utilisé pour configurer le circuit programmable, soit à l'aide d'une mémoire non-volatile associée au circuit, soit directement à partir d'une interface externe (processeur, bus, liaison parallèle, etc…).

2.3 Les outils de développement :

L'électronicien a toujours utilisé des outils de description pour représenter des structures logiques ou analogiques. Le schéma structurel que l'on utilise depuis si longtemps et si souvent n'est en fait qu'un outil de description graphique. Aujourd'hui, l'électronique numérique est de plus en plus présente et tend bien souvent à remplacer les structures analogiques utilisées jusqu'à présent. Ainsi, l'ampleur des fonctions numériques à réaliser nous impose l'utilisation d'un autre outil de description **[BLOTIN]**.

Avec l'apparition des circuits logiques programmables de type SPLD, CPLD ou FPGA, l'utilisateur peut créer dans ces derniers toutes les fonctions logiques en utilisant des outils de développement mis à sa disposition par leurs fabricants ; ces outils doivent permettre de passer de la description du comportement d'une fonction logique à son câblage dans le circuit et cela de la manière la plus simple possible (Figure 2.4).

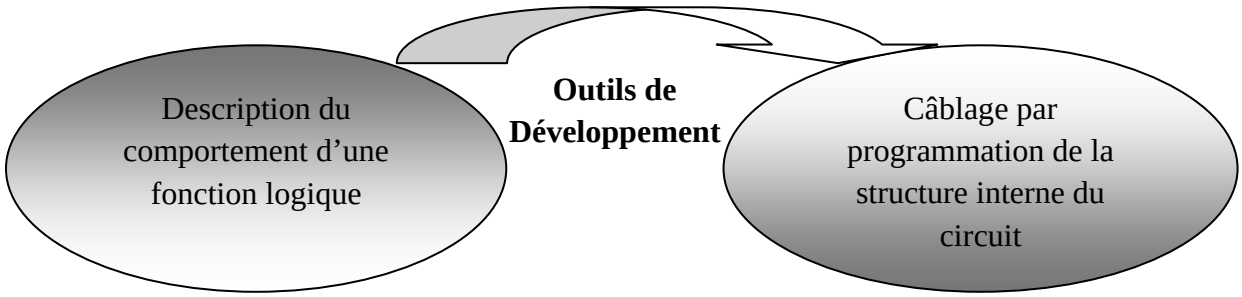


Figure 2.4 : L'utilité d'un outil de développement.

2.3.1 Les outils de CAO (Conception Assistée par Ordinateur) :

La conception assistée par ordinateur CAO (*Electronic Design Automation* : EDA) est la catégorie des outils servant à la production des systèmes électroniques allant des circuits imprimés jusqu'aux circuits intégrés **[Wikp.com]**.

Avec la miniaturisation continuelle de la technologie des semi-conducteurs, les circuits électroniques sont devenus trop complexes pour être conçus à la main ce qui a résulté la nécessité des outils informatiques adéquats et c'était la naissance des outils de CAO.

Les concepteurs réalisent et testent les circuits sur ordinateur avant de lancer la fabrication On peut distinguer trois types d'outils de CAO selon les modes de descriptions trouvés dans ces derniers :

- Un outil de CAO utilisé que pour la synthèse et la simulation (comme MODELSIM).
- Un outil de CAO qui utilise le mode schématique et textuel (comme ISE de XILINX).
- Un outil de CAO qui peut utiliser le mode schématique, textuel et permet même le dessin du masque (*layout*) (comme CADENCE).

La CAO est divisée en plusieurs sous-étapes. Elles s'alignent la plupart du temps avec le processus de fabrication du concept pour masquer la création **[KARABERNOU-09]**. Ce qui suit s'applique à la création de circuit/ASIC/FPGA mais cette méthodologie est similaire à celle appliquée pour les circuits conventionnels :

2.3.1.1 Les outils de synthèse et simulation :

L'outil de synthèse (Figure 2.5) a pour objectifs : minimiser la surface de silicium, le temps de propagation ainsi que la consommation.

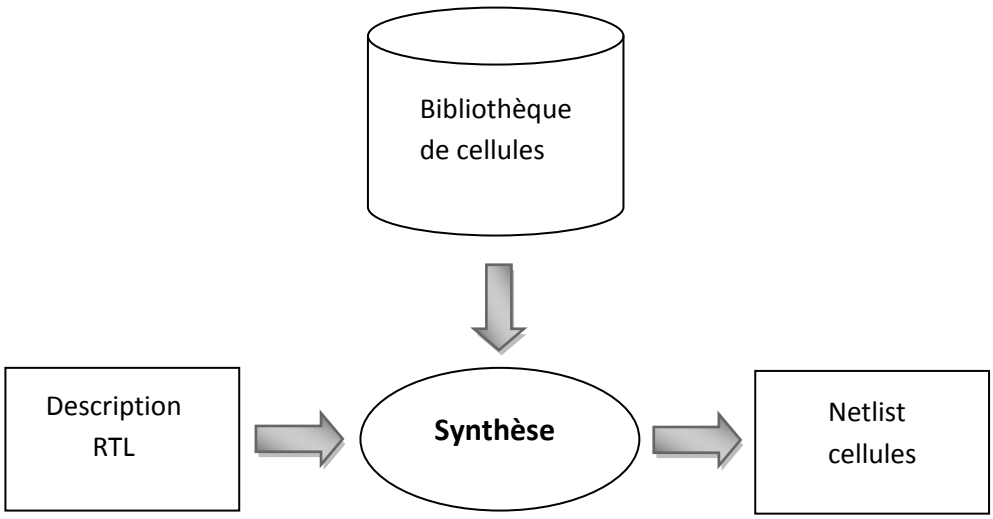


Figure 2.5 : L'outil de synthèse.

2.3.1.2 Les outils de placement et routage :

Les outils de placement et routage (Figure 2.6) permettent de minimiser la surface globale du silicium et les longueurs d'interconnections.

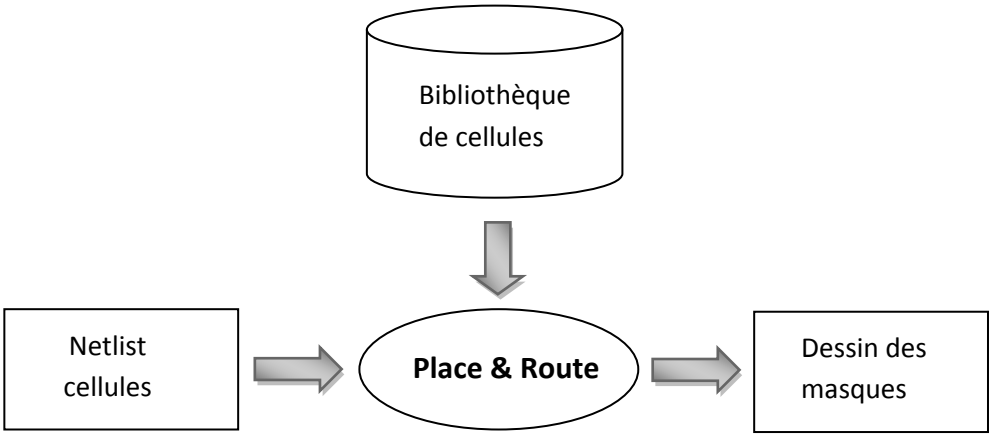


Figure 2.6 : Les outils de placement et routage.

On peut résumer les plus grandes entreprises de CAO électroniques dans le tableau suivant (Tableau 2.1) **[Wikp.com]**:

Entreprise	Localisation	Capitalisation (09/2009)
Synopsys	Mountain View, Californie	$3230 millions
Cadence Design Systems	San Jose, Californie	$1890 millions
Mentor Graphics	Wilsonville, Oregon	$919 millions
Zuken Inc.	Yokohama, Japon	$187 millions
Magma Design Automation Inc	Santa Clara, Californie	$75 millions
Altium Ltd.	Sydney, Australie	

Tableau 2.1 : Les plus grandes entreprises de CAO électroniques.

2.3.2 Les différentes approches de description d'un circuit :

2.3.2.1 Description physique (Dessin au micron) :

Dans la première époque, la conception de circuits intégrés était un métier manuel. On dessinait les composants (transistors) à la main, sur un papier spécial (*Mylar*) avec des crayons de couleur. C'est ce qu'on appelle le ***dessin au micron***.

Une telle technique limitait naturellement la complexité des dispositifs conçus, mais avec l'apparition des différents outils de CAO tel que CADENCE, le dessin des circuits complexes est devenu plus facile (Figure 2.7).

Le dessin des masques (*Layout*) doit être fait de façon à assurer que les structures dessinées fonctionnent correctement après fabrication. Pour cela, il ya un certain nombre de contraintes à respecter lors du dessin : les règles de dessin. Celles-ci dépendent fortement du processus de fabrication et varient d'un fabricant à l'autre.

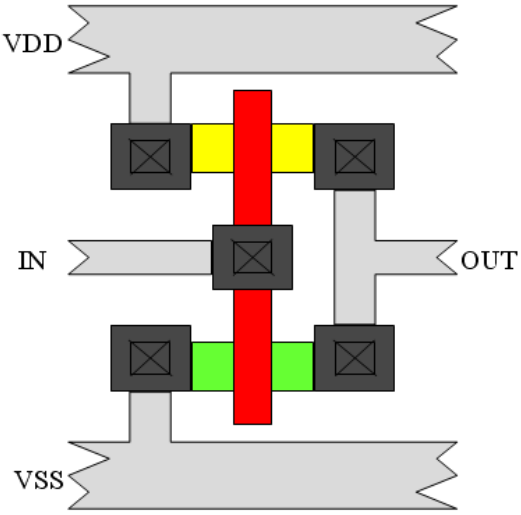


Figure 2.7 : Dessin au micron d'un inverseur CMOS.

2.3.2.2 Description schématique :

L'apparition des interfaces graphiques et donc des éditeurs de schémas font la naissance d'une nouvelle description dans la réalisation des circuits qui est le schématique.

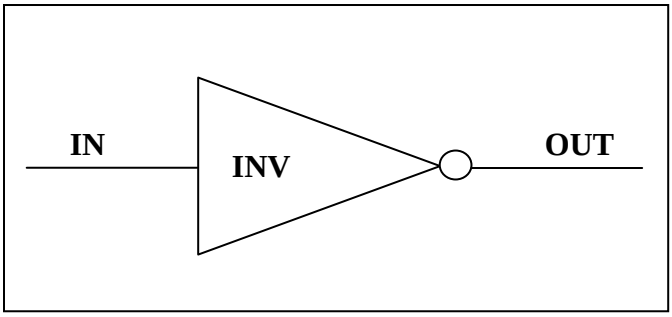


Figure 2.8 : Description schématique de l'inverseur CMOS.

On peut comprendre facilement la description schématique du composant (Figure 2.8) que son dessin de masque (Figure 2.7).

2.3.2.3 Description textuelle :

Grâce aux nouvelles possibilités de description à un niveau d'abstraction plus élevé, les langages *fonctionnels* (ou *comportementaux)* de description de matériel (Figure 2.9) ont répondu à des besoins fondamentaux des concepteurs de circuits intégrés :

- La réduction des temps de conception.
- L'accélération des simulations qui devenaient prohibitives avec l'accroissement de complexité des dispositifs.
- La normalisation des échanges : le monde de l'électronique a souffert pendant longtemps de la multiplicité des langages, des formats, des outils de CAO utilisés.

L'émergence des langages fonctionnels de description de matériel a été l'occasion de remédier à ce problème. Désormais, des langages normalisés et universellement reconnus servent aux échanges entre partenaires industriels, entre fournisseurs et clients. VERILOG, VHDL et SystemC en sont des exemples (cf. § 2.3.3).

- La fiabilité : les langages HDL sont conçus pour limiter en principe les risques d'erreur.
- La portabilité : les langages normalisés sont très largement portables.
- La réutilisabilité : les modifications et adaptations sont rendues plus simples donc moins risquées et moins coûteuses.

```
Entity INV is
port (IN : in STD_LOGIC;
OUT : out STD_LOGIC);
end INV;

Architecture INV of INV is
begin
OUT <= not (IN);
end INV;
```

Figure 2.9 : Description textuelle d'un inverseur CMOS (en VHDL).

2.3.3 Les langages de description :

Dans les années 80, des langages plus évolués appelés langages de description matérielle ou HDL (***H**ardware **D**escription **L**angage)* sont apparus. Avec ce type de langage, il est possible de décrire un système séquentiel par la description de la machine à états (méthodes graphiques ou tabloïdes). Les principaux langages disponibles sont résumés dans le tableau 2.2. Il s'agit de langages de synthèse logique, de programmation ou de description matérielle. Ils peuvent être de type propriétaire comme ABEL, CUPL ou PALASM ou normalisé tels que VHDL [IEEE 87 93] ou VERILOG [IEEE 95 01].

Langages	Sociétés
ABEL	Xilinx (*Data I/O Corporation*)
CUPL	*Logic Devices, Inc.*
PALASM	*Lattice* (MMI/AMD).
VERILOG	*Verilog* et norme IEEE.
VHDL	Norme IEEE.
HANDEL-C	CELOXICA
L'outil Stream-C	*Los Alamos National Laboratories*
SystemC	*Synopsys - CoWare*

Tableau 2.2 : Quelques langages de description.

Une des particularités de ces langages, est qu'ils permettent de dissocier la spécification du comportement des composants de leur réalisation. Le concepteur peut tout d'abord valider la spécification fonctionnelle du système, en modélisant le comportement de ses composants à l'aide des constructions de haut niveau fournies par le langage (on parle alors de spécification comportementale), pour ensuite s'intéresser à leur réalisation. Il spécifie alors leur architecture à l'aide d'un sous-ensemble de ce même langage (on parle alors de description RTL). Cette différentiation permet ainsi au concepteur d'explorer les différents choix architecturaux permettant la réalisation d'une même fonctionnalité.

2.3.3.1 CUPL :

Le premier langage de programmation qui est été introduits avec l'apparition des circuits programmables, le CUPL. Il permettait d'écrire des équations booléennes et des tables de vérité. Cette 1ère étape essentielle entre la table de fusibles et la programmation d'équations a été très importante pour le développement qui a suivi : elle consiste à passer d'une programmation très proche du matériel à une programmation *software*, avec toutes les possibilités qui s'ouvrent dès lors. L'une de celles-ci est la simulation. Bien que les premières moutures du CUPL fussent fournies sans simulateur, la programmation de manière logicielle laissait entrevoir cette possibilité et très rapidement, les avantages d'une simulation avant le test physique se sont imposés. La répartition des temps de développement n'est depuis ce moment plus la même, elle augmente à la conception et à la simulation, alors qu'elle diminue à l'écriture du code source et au dépannage **[GUEX-98]**.

2.3.3.2 ABEL (*Advanced Boolean Equation Logical):*

ABEL est un langage de description hardware développé par DATA CORPORATION pour les dispositifs de logique programmables. Il est également utilisé pour décrire le comportement d'un système logique combinatoire et séquentiel dans des formes rencontrées en algèbre de Boole. Ce support présente de manière concise les options et la syntaxe d'ABEL. C'est un langage plus simple que VHDL, il permet d'écrire des structures indépendantes du choix technologique final, qui sera fait après simulation et optimisation (avec généralement une aide par les outils de CAO).

2.3.3.3 VHDL (*VHSIC (Very High Speed Integrated Circuit) Hardware Description Language*):

Développé dans les années 80 aux États-Unis, le langage VHDL est ensuite devenu une norme IEEE numéro 1076 en 1987. Révisée en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité du langage, cette norme est vite devenue un standard en matière d'outils de description de fonctions logiques **[BLOTIN]**. A ce jour, on utilise le langage VHDL pour :

- concevoir des ASIC,
- programmer des composants programmables du type PLD, CPLD et FPGA,
- concevoir des modèles de simulations numériques ou des bancs de tests.

Le langage VHDL assure la portabilité des conceptions, permet de synthétiser des fonctions logiques complexes à partir d'une description concise. Il est écrit indépendamment d'une technologie particulière ou d'une chaîne de conception, facilite le passage entre les différentes technologies utilisables qui évoluent sans cesse.

Le VHDL a pour autre avantage la possibilité de décrire des systèmes très complexes en quelques lignes de code. Cependant, la synthèse de ces fichiers devient de plus en plus difficile et un code VHDL doit être optimisé pour un circuit cible. Dans la pratique, une fois ces modules développés, on les intègre dans une bibliothèque et les réutilise en cas de besoin ultérieur. Pour des circuits plus importants, cela peut devenir très compliqué à développer, surtout pour des entreprises de petite à moyenne importance. Certaines entreprises de circuits ont alors proposé de vendre leur propriété intellectuelle sous forme de code VHDL et de *tests-benchs* prêts à l'emploi. L'utilisateur n'a alors qu'à assembler des modules sans avoir à en écrire le code interne. Dans un proche avenir, il sera même possible, à l'échelle industrielle, de faire du *codesign*, c'est-à-dire de développer un système et de choisir ensuite ce qui sera implanté sous forme matérielle et ce qui le sera sous forme logicielle **[GUEX-98]**.

La description VHDL peut prendre l'un des trois types de modèles :
- Le modèle comportemental qui décrit la fonctionnalité d'un objet par un algorithme séquentiel ou une table de vérité, sans référence à une implémentation quelconque.
- Le modèle flux de données qui décrit le flux entre l'entrée et la sortie au niveau bit, par des équations élémentaires.
- Le modèle structurel qui représente la constitution de l'objet en un ensemble d'objets élémentaires interconnectés (proche du schéma).

2.3.3.4 VERILOG :

VERILOG est un langage de description de matériel, c'est-à-dire un langage utilisé pour décrire un système numérique matériel. Il a été inventé par *Gateway Design Automation Inc* en 1985, et en 1990, la société de CAO Cadence rachète Gateway. Elle décide de rendre les spécifications de VERILOG publiques, qui deviennent un standard IEEE en 1995 (IEEE Std. 1364-1995).

L'IEEE à créer une nouvelle version du langage, VERILOG 2001 puis VERILOG 2005 à cause des lacunes de VERILOG 1995. Ces versions du langage sont maintenant acceptées par la quasi-totalité des outils de l'industrie.

VERILOG peut modéliser un système par n'importe quelle vue, structurelle ou comportementale, à tous les niveaux de description, de plus il peut servir non seulement à simuler un système mais aussi à le synthétiser, c'est-à-dire être transformé par des logiciels adaptés (synthétiseurs) en une série de portes logiques prêtes à être gravées sur du silicium.

VERILOG est l'un des trois grands langages de description de matériel utilisés dans l'industrie, avec VHDL et SystemC. Il combine deux aspects :

- la simulation : il permet de décrire l'enchaînement d'événements ;

- description par combinaison d'éléments (modules, expressions, portes logiques…), ce qui permet de synthétiser des circuits.

La structure du langage VERILOG permet de décrire les entrées et les sorties de modules électroniques, pour définir des portes logiques virtuelles. La combinaison de modules permet de réaliser des schémas électroniques virtuels complexes qu'il est alors possible de tester dans un programme de simulation.

2.3.3.5 HANDEL-C :

Le langage HANDEL-C a été conçu aux États-Unis *(Oxford University Hardware Compilation Group*) par la société CELOXICA. L'objectif de ce langage est de permettre au concepteur un prototypage rapide d'applications sur cible FPGA à partir de spécifications dans un langage de haut niveau, le HANDEL-C est un dérivé du langage C.
Le principe est le suivant : lors de la phase de compilation du programme à implanter sur le circuit reconfigurable, chaque instruction du programme source est transformée en une entité matérielle qui effectue le ou les traitements associé(s) à cette instruction. Un ensemble de blocs matériels est ainsi généré, représentant l'ensemble des instructions du programme. Celles-ci sont alors exécutées de façon totalement séquentielle à l'aide d'un contrôleur à jeton qui, à chaque cycle d'horloge, active le bloc instruction courant.

En conséquence, et afin de permettre au concepteur d'exploiter tout le parallélisme disponible sur un circuit FPGA, le langage permet d'exploiter la concurrence entre différentes parties du programme. Cette concurrence, qui permet l'exécution en parallèle de plusieurs instructions (ou flot d'instructions) doit cependant être explicite dans le programme, elle s'exprime à l'aide d'une construction propre au langage **[DERRIEN-02]**.

2.3.3.6 L'outil Stream-C :

L'outil Stream-C est basé sur une spécification de l'application en un sous-ensemble du langage C, il est développé au sein du *Los Alamos National Laboratories*.
L'objectif de Stream-C est de spécifier le comportement de l'architecture à l'aide de constructions de haut-niveau qui permettent de modéliser l'application comme un ensemble de processus concurrents. Ceux-ci communiquent entre eux à l'aide de tubes de données (*stream*s) en se basant sur un protocole de communication de type producteur/consommateur non bloquant (le modèle de calcul se rapproche beaucoup d'un formalisme de type CSP).

Dans l'outil Stream-C, le parallélisme est donc exprimé de façon explicite : il apparaît dans la concurrence entre les processus formant le programme. Lors de la phase de *compilation* de la spécification en C, chaque tâche est soit transformée en une entité matérielle autonome (on génère alors une description en VHDL synthétisable de l'architecture associée au processus), soit compilée pour être exécutée comme un simple processus sur le processeur hôte de la ressource reconfigurable **[DERRIEN-02]**.

2.3.3.7 Le systemC :

SystemC est, comme VERILOG et VHDL, un langage de description de matériel. C'est le fruit de contributions de plusieurs sociétés. En 1989, *Synopsys* met son outil commercial SCENIC dans le domaine libre, et crée la version 0.9 de SystemC. Une première contribution de *Frontier Design* donne lieu à la version 1.0, et une autre de *CoWare* aboutit en 2000 à la version 1.1, première version officielle de SystemC. L'OSCI (*Open SystemC Initiative*) est alors crée, rassemblant une multitude de sociétés et laboratoires de recherche. Cette organisation est en charge de diffuser, promouvoir et rédiger les spécifications de SystemC.

Les spécifications de SystemC ont été étendues en 2001 à la modélisation de systèmes abstraits (de très haut niveau, avant partitionnement matériel / logiciel), aboutissant à la version 2.0. En 2005, l'IEEE a approuvé une version de référence de SystemC, appelée IEEE1666-2005 et correspondant à la version 2.2.0 de SystemC, qui est l'actuelle version stable.

SystemC a pour objectif de modéliser des systèmes numériques *matériels et logiciels* à l'aide de C++. Il permet donc de modéliser non seulement des systèmes *matériels*, mais aussi des systèmes *logiciels, mixtes ou non-partitionnés* (où on ne sait pas encore ce qui sera fait en logiciel, et ce qui sera fait en matériel) **[Wikp.com]**.

2.4 Conclusion :

Dans ce chapitre, on a pu exposer les différentes méthodologies de conception qui permettent le développement de systèmes numériques complexes sur puce.

L'approche du flot de conception est utilisée afin de faciliter la conception des circuits logiques à hautes densités (les circuits complexes). Cette architecture matérielle complexe nécessite l'usage des outils de conception assistée par ordinateur (CAO) capables de gérer les différentes étapes de production d'un circuit logique programmable complexe, tout en tenant compte des critères de performance et de superviser l'exécution des applications et les contraintes de cahier des charges (surface, temps, consommation…).

L'ensemble des démarches du flot de conception n'est pas toujours respecté. En particulier, la modélisation comportementale n'est pas toujours réalisée ce qui permet de diminuer le temps de conception mais au prix d'un risque important d'erreurs qui, détectées trop tard, peuvent remettre en cause la faisabilité globale du projet.

Les outils présentés dans cette section permettent aussi la conception rapide d'une architecture matérielle sur un circuit reconfigurable, à partir d'une spécification comportementale. Donc nous allons exhiber dans le chapitre 3 l'utilité de suivre une méthodologie de conception pour réaliser une architecture matérielle sur le circuit reconfigurable FPGA. Nous allons nous intéresser à l'un des outils de CAO (le ISE de XILINX), capable de synthétiser, faire le placement-routage et générer le fichier de

configuration, puis télécharger ce dernier dans le FPGA par le port parallèle ou bien un port USB (cet outil est compatible avec la carte de développement utilisée).

Chapitre 3

Etude et implantation d'un contrôleur vidéo sur un circuit reconfigurable de type FPGA

3.1 Introduction :

Dans le cadre des recherches effectuées sur les circuits logiques programmables ainsi que sur leurs méthodologies de conception, nous avons choisis la conception d'un contrôleur vidéo en technologie FPGA.

Le cahier des charges du projet consiste à concevoir puis implémenter une architecture matérielle sur une carte numérique à base de FPGA, permettant la conception d'un jeu vidéo qui sera affiché sur un moniteur VGA. L'architecture étudiée s'appuie sur des contraintes d'encombrement minimales et de souplesse maximale.

Dans ce chapitre, nous vous présentons dans un premier temps l'approche globale de conception en exposant les différents outils utilisés pour réaliser le projet. Ensuite, en se basant sur le schéma de notre architecture, nous détaillerons les différents blocs et mettrons leurs rôles en évidence. Puis une étude du résultat final est proposée. Et à la fin, une brève conclusion rappelle les différents enjeux du projet.

3.2 Présentation de l'environnement :

Le développement d'architectures à l'aide de FPGA nécessite l'emploi d'outils de développement. Cet outil comporte, en général, un logiciel de conception, une carte de développement pour la vérification du fonctionnement électrique de l'architecture conçue, et un micro- ordinateur hôte pour la gestion de l'ensemble.

Le système réalisé (Figure 3.1) est composé :

- ✓ D'un micro-ordinateur doté d'un logiciel de développement (ISE 9.2i de XILINX) permettant la conception de l'architecture matérielle et de son implantation dans le FPGA.
- ✓ Une carte de développement (NEXYS2 de XILINX) qui contient le circuit reconfigurable FPGA (SPARTAN 3E 1200K).
- ✓ Un moniteur VGA pour l'affichage de nos résultats (l'image affichée a une résolution de 640x480 en 256 couleurs).

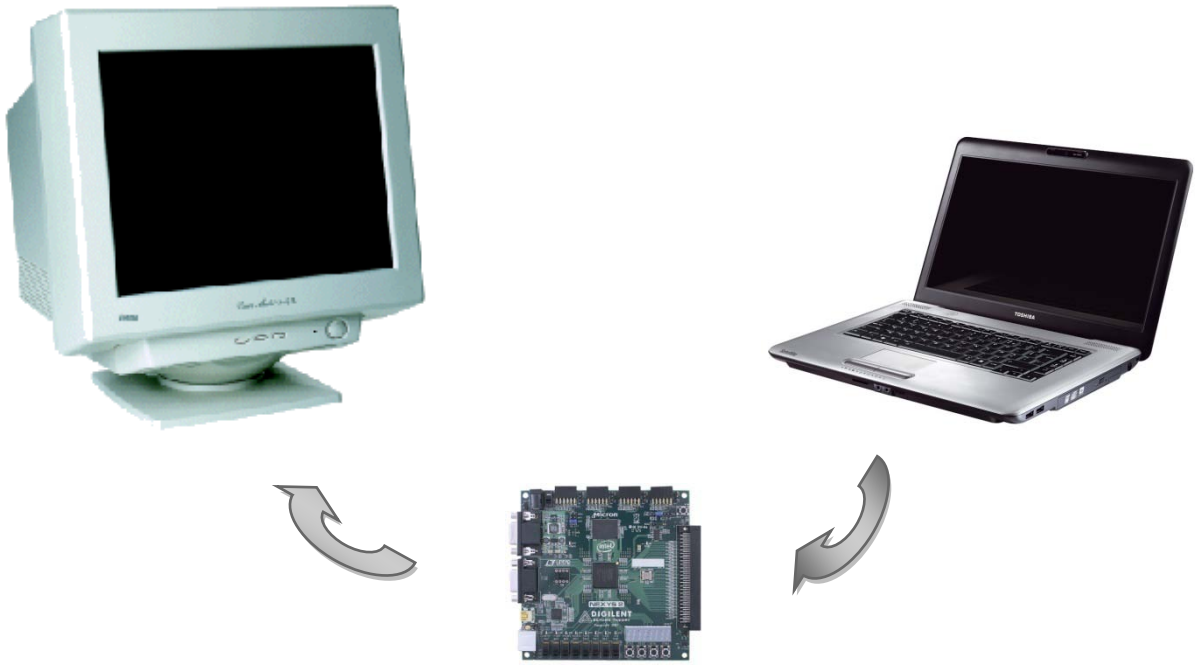


Figure 3.1 : Chaîne de développement

3.2.1 Le logiciel de développement :

ISE 9.2i est un outil de CAO fourni par la société XILINX **[Xil.com]** permettant la gestion complète du flot de conception et implantation d'architecture numérique sur FPGA. L'interface graphique d'ISE 9.2i est présentée par la figure ci-dessous (Figure 3.2).

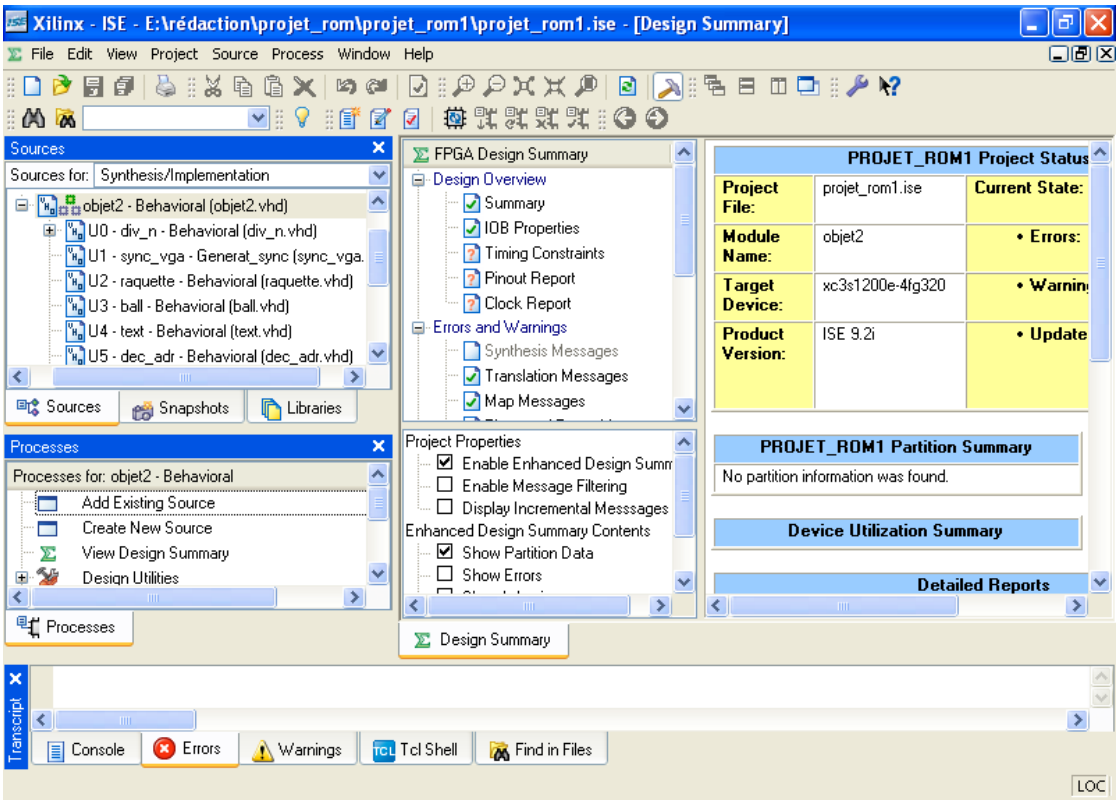


Figure 3.2 : Environnement de développement ISE 9.2i

Ce logiciel est une suite d'outils qui permet de faire la saisie des spécifications (en schématique et/ou en textuelle VHDL/VERILOG) d'architectures numériques, de procéder aux différentes vérifications, de réaliser des simulations, de faire la synthèse, le partitionnement, le placement et le routage et enfin la génération du fichier de configuration, qui sera implémenté sur le circuit reconfigurable FPGA de XILINX à l'aide d'une liaison JTAG ou une liaison série USB. La procédure d'utilisation de ces outils de CAO est présentée dans l'annexe A.

3.2.2 La carte de développement :

Nous utiliserons une carte de développement NEXYS2 à base de FPGA SPARTAN3E XC3S1200 qui intègre l'équivalent de 1200 000 portes cadencé à 50MHz (Figure 3.3). Cette carte est constituée du FPGA, de la mémoire (SDRAM, FLASH), d'un ensemble de périphériques d'entrées-sorties et d'autres éléments nécessaires au fonctionnement de la carte. L'architecture globale de la carte est présentée en annexe B.

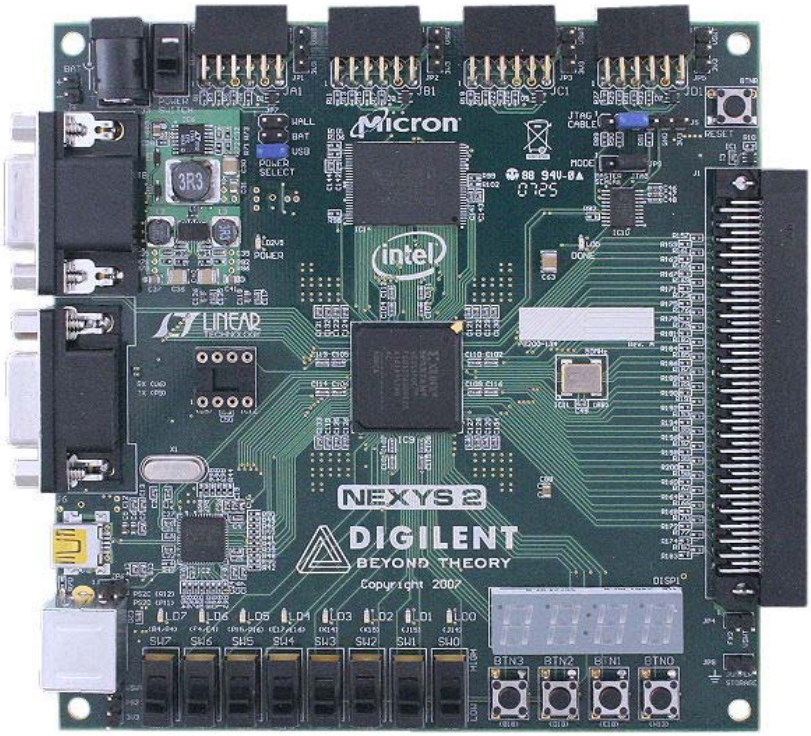


Figure 3.3 : Carte NEXYS2 de XILINX.

La carte NEXYS2 est une plateforme complète prête à l'emploi, voici la description des principaux éléments de la carte (on trouve dans l'annexe B les descriptions schématiques de chaque élément):

- ✓ **FPGA** (SPARTAN 3E XC3S1200) : comporte 1200K portes.
- ✓ **FLASH** (STRATAFLASH d'Intel) : une mémoire non volatile de 16 Mbits.
- ✓ **SDRAM** : une mémoire vive de 16 Mbits.
- ✓ **Oscillateur** (50 MHz) : fournit le signal d'horloge au circuit FPGA.
- ✓ **Interface :**

- **Afficheurs** (7 segments) **:** permet l'affichage de 7 segments à 4 chiffres.
- **LED** : la carte possède 8 électroluminescentes.
- **Boutons poussoirs** : la carte possède 4 boutons.
- **Contacts coulissants :** la carte possède aussi 8 contacts (*Switches*).
- **Port PS/2 :** un port pour l'utilisation d'une souris ou un clavier.
- **Connecteur VGA** : de type DB15, il permet la connexion à un moniteur VGA.
- **Connecteur RS232** : de type DB9, il permet le raccordement à un port série.
- Port **USB2** : permet la Configuration du FPGA avec une vitesse élevée de transfert des données.
- **Autres connecteurs périphériques** : connecteurs d'expansion (HIROSE FX2).

3.2.3 Le moniteur VGA (*Video Graphics Adapter*) :

VGA est une norme de signal vidéo que l'on trouve principalement dans les ordinateurs individuels. Ce signal vidéo est constitué de 5 signaux, 3 analogiques avec des niveaux de 0,7 à 1,0 volt pour chacune des couleurs Rouge, Vert et Bleu «RVB» (RGB : *Red*, *Green*, *Blue*) accompagnés de 2 signaux logiques que sont les synchronisations horizontale et verticale.

Le principal composant du moniteur est le tube cathodique dont l'écran est la face visible pour l'utilisateur. Le faisceau d'électrons est dévié selon un parcours horizontal de gauche à droite de l'écran, puis retour à la ligne avec décalage vers le bas et ceci jusqu'à atteindre le point le plus bas à droite (Figure 3.4). Dans ce parcours, l'information RVB est utilisée pour contrôler l'intensité du faisceau d'électrons. Cela donnera selon le cas un point de couleur rouge, bleu ou vert (couleurs primaires) plus ou moins intenses, l'œil humain percevant une couleur résultante de ces trois points très proches physiquement **[BOUKHLIF-94]**.

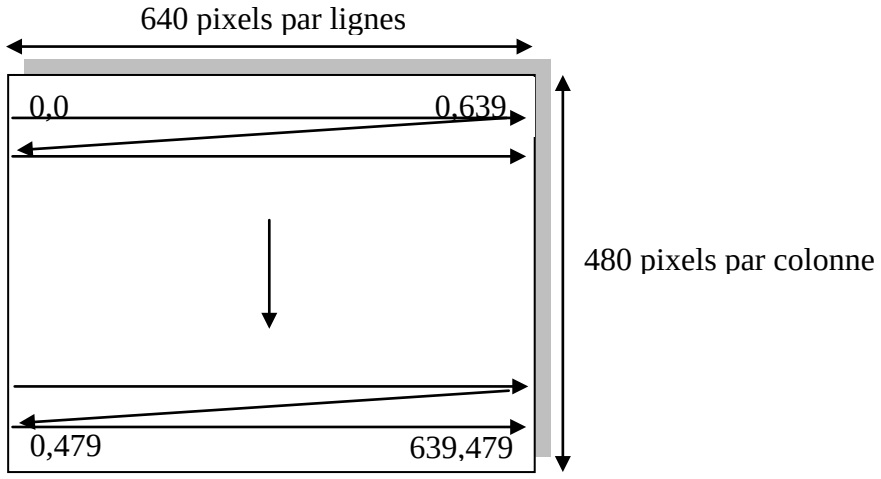


Figure 3.4 : Image de 640 x 480.

En VGA standard, l'écran contient 640 par 480 pixels (*picture element*) qui doivent être balayés à une fréquence supérieure à 30 Hz si l'on veut éviter un effet de scintillement. On

trouve en général sur les PC une fréquence de rafraîchissement de 60 Hz (60 images par seconde). Les moniteurs du commerce fonctionnent tous dans une certaine plage autour de cette fréquence de référence. Si l'on veut évaluer approximativement la fréquence pixel, il suffit de diviser 1/60 par le produit 680x480. Cela donne 50 ns environ entre 2 pixels. Bien que l'horloge de la carte NEXYS2 soit à 50 MHz, on fera les calculs de timing à 25 Mhz, le moniteur s'accordant sur la fréquence qu'il recevra.

Dans notre cas, la trame VGA (Figure 3.5) étant découpée en 521 lignes une fréquence image de 60 Hertz. Une ligne correspond pour une fréquence pixel de 25 MHz à un comptage de 800 points.
Le balayage vertical se décompose en une synchro (équivalente à 2 lignes) suivi de 29 lignes noires suivis des 480 lignes de pixels suivies de 10 lignes noires (2+29+480+10=521).
Le balayage horizontal se décompose en un signal de synchronisation (équivalent à 96 points) un pallier avant noir de 48 points (*Back porch*), suivi de 640 pixels et un pallier arrière de 16 points (*Front porch*) (96+48+640+16=800).

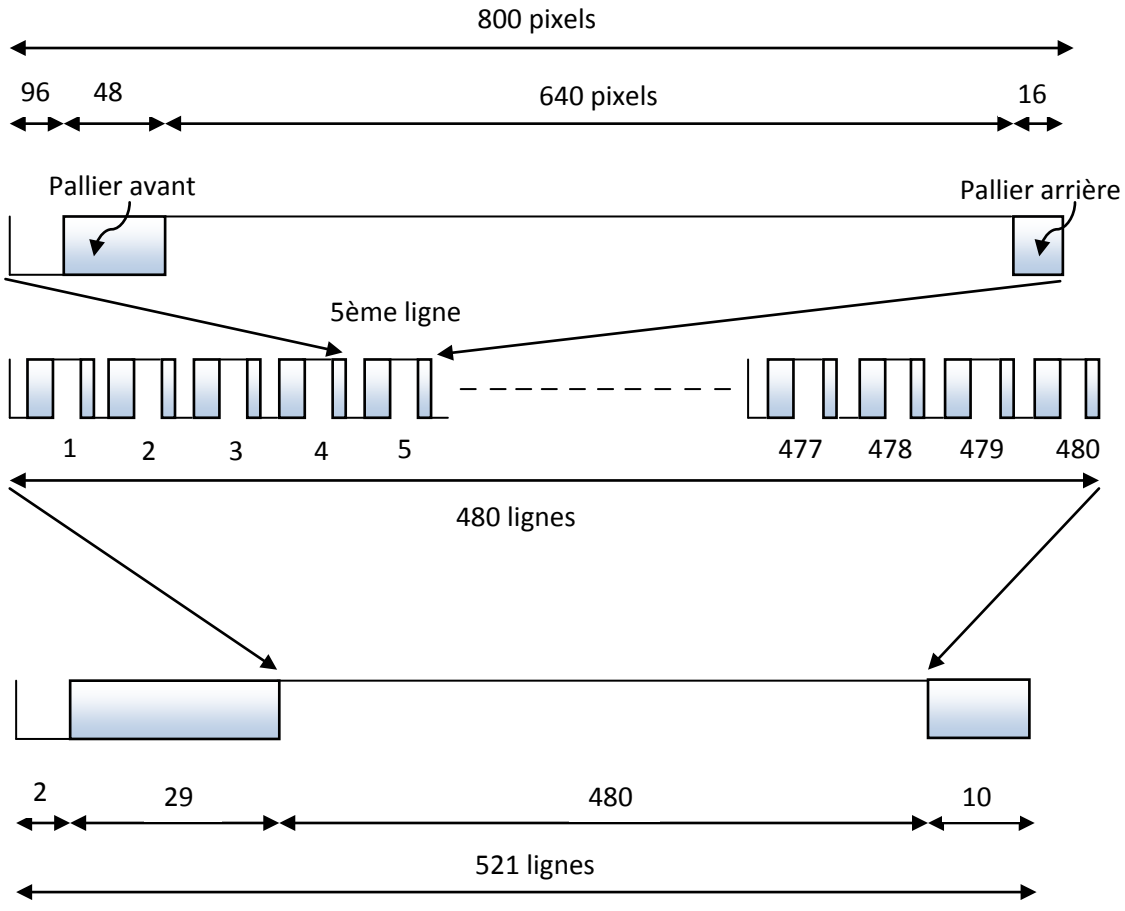


Figure 3.5 : Le timing des deux synchronisations horizontale et verticale.

Les valeurs des temps de synchronisation et d'affichage sont données par le tableau 3.1 :

Symbole	Paramètre	Synchro Verticale			Synchro Horizontale	
		Temps	Horloge	Ligne	Temps	Horloge
T_S	Durée synchro	16.7 ms	416800 cycles	521	32 µs	800 cycles
T_{DISP}	Durée affichage	15.36 ms	384000 cycles	480	25.6 µs	640 cycles
T_{PW}	Largeur pulsation	64 µs	1600 cycles	2	3.84 µs	96 cycles
T_{FP}	Noir avant pulsation	320 µs	8000 cycles	10	640 ns	16 cycles
T_{BP}	Noir après pulsation	928 µs	23200 cycles	29	1.92 µs	48 cycles

Tableau 3.1 : Les durées de synchronisation.

3.3 Description matérielle du système:

3.3.1 Objectifs et définition du cahier des charges :

L'objectif de ce projet est de valider la méthodologie de conception décrite dans le chapitre précédant (cf. § 2.2.2), à travers une application qui consiste à mettre en œuvre un jeu vidéo. Ce dernier sera affiché sur un écran VGA de résolution 640x480 à 60 Hz. L'application une fois conçue, la saisie des spécifications sera réalisée à l'aide d'un langage de description de matériel (VHDL) puis implémentée sur un circuit reconfigurable FPGA.

Dans ce jeu on dispose d'une balle et d'une raquette apparaissant sur un écran vidéo. La balle se déplace avec un mouvement rectiligne uniforme dans un espace à deux dimensions délimité par quatre bords, deux verticaux (gauche et droit) et deux horizontaux (haut et bas). Cet espace est matérialisé par les dimensions d'un écran d'ordinateur (Figure 3.6) La raquette se déplace uniquement sur le côté bas sous l'action du joueur (à l'aide des boutons poussoirs de la carte de développement). La balle peut rebondir sans perte de vitesse en respectant les lois de la dynamique sur les côtés haut, droit et gauche et sur la raquette. Le jeu se termine lorsque la balle touche le côté bas.

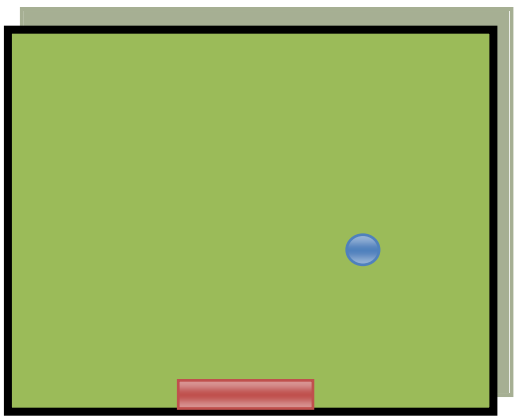


Figure 3.6 : Jeu vidéo.

3.3.2 Les spécifications fonctionnelles :

Pour afficher un objet (une image) sur un écran de type VGA, il faut générer deux signaux de synchronisation horizontale et verticale «SYNC_H, SYNC_V» (déflexion) et les trois composantes Rouge, Vert et Bleu « R, V et B » correspondantes à l'information visuelle.

Chaque objet est caractérisé par sa forme et ses coordonnées dans le plan image. Les coordonnées de la raquette sont fournies par deux boutons poussoirs, un pour le déplacement à droite et l'autre pour le déplacement à gauche de la raquette. Les coordonnées de la balle sont calculées en fonction de la position actuelle et d'un algorithme décrivant la trajectoire.

Les signaux de sortie (SYNC_H, SYNC_V, R, V et B) respectent les spécifications du standard type pour une image de 640 pixels par 480 lignes. Dans cette image on distingue deux objets la balle (forme ronde d'une couleur bleu) et la raquette (forme rectangulaire d'une couleur rouge) sur un fond d'une couleur verte. Ces objets sont repérés dans l'image par les coordonnées (x, y) de leurs points extrêmes haut et gauche : (a, b) et (c, d) de la raquette et de la balle respectivement (Figure 3.7). (Lr, Lb) et (Hr, Hb) représentent respectivement les longueurs et hauteurs du motif raquette et du motif balle.

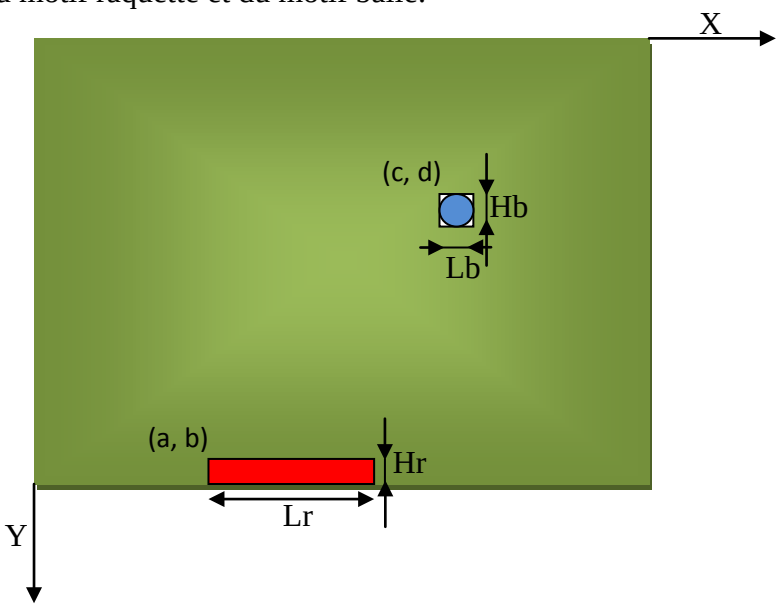


Figure 3.7 : Localisation des objets dans l'image.

Le système devra générer les cinq signaux décrits dans le paragraphe précédent, de façon à faire apparaître en temps réel la balle et la raquette (Figure 3.8).

Le mouvement de la balle est caractérisé par celui de sont point de repère. A tout instant l'équation du déplacement sur la trajectoire considérée est de type :

$\boldsymbol{d(t) = V \times t}$ Où : **V** représente la vitesse de la balle et **t** le temps.

Le changement de direction de la balle se fait en respectant les lois de la dynamique et en considérant des chocs élastiques et sans perte de vitesse. Les rebonds sont autorisés sur les côtés verticaux, le côté horizontal haut et sur la raquette.

La raquette se déplace le long du bord horizontal bas de l'image. L'équation du mouvement s'écrit :

$$Xr\,(t) = \begin{cases} (D - G) \times V \times t + Xr_0 & \textbf{Pour } 0 < Xr\,(t) < Xrmax \\ Xr_0 & \textbf{Pour les autres cas} \end{cases}$$

Où **Xr (t)** : l'abscisse du repère de la raquette à l'instant **t**.

V : vitesse de déplacement au choix du concepteur.

G, D : commande de déplacement respectivement à gauche **(G=1)** et à droite **(D=1)**. Pas de déplacement si les deux signaux sont en même temps à **1** ou à **0**.

Xr_0 : position précédente du repère de la raquette.

Xrmax : 640-longueur raquette.

Vue externe du système :

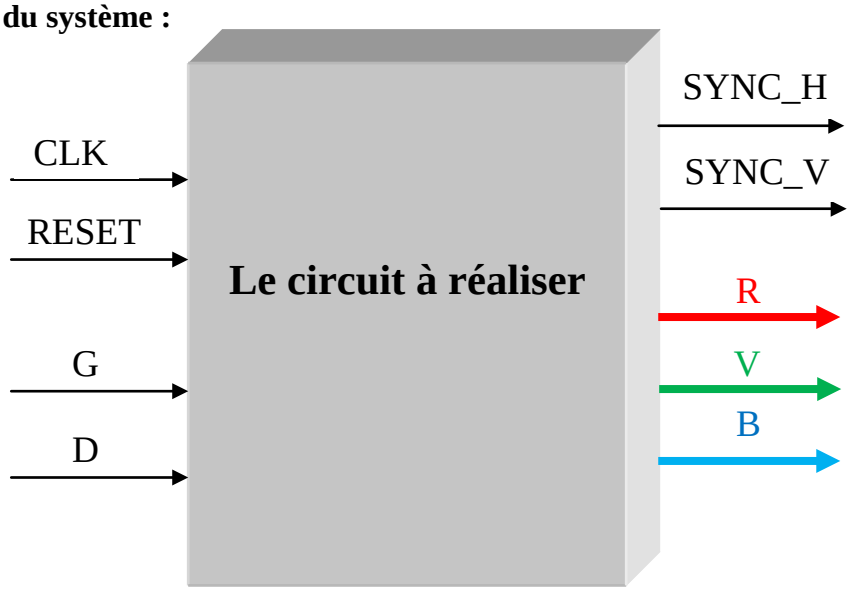


Figure 3.8 : Architecture externe du circuit.

Le système à réaliser a pour signaux d'entrées :

- CLK : horloge principale fournie par la carte NEXYS2 de 50 MHZ.
- RESET : signal de remise à zéro du système et de relance du jeu. Ce signal est fourni par le premier bouton poussoir de la carte NEXYS2.
- D : le niveau actif de ce signal permet de déplacer la raquette vers la droite. Ce signal est fourni par le deuxième bouton poussoir de la carte NEXYS2.
- G : le niveau actif de ce signal permet de déplacer la raquette vers la gauche. Ce signal est fourni par le troisième bouton poussoir de la carte NEXYS2.

Les signaux de sorties du système se résument en :

- SYNC_H : signal de synchronisation horizontale.
- SYNC_V : signal de synchronisation verticale.
- R : la composante rouge de l'image couleur.

- V : la composante verte de l'image couleur.
- B : la composante bleu de l'image couleur.

R, V et B constituent les trois composantes Rouge, Vert et Bleu de l'image couleur. Dans notre cas, les deux composantes R et V sont codées sur 3 bits chacun et la composante B est codée sur 2 bits pour avoir 256 couleurs (Figure 3.9). Un pixel est donc codé sur un octet.

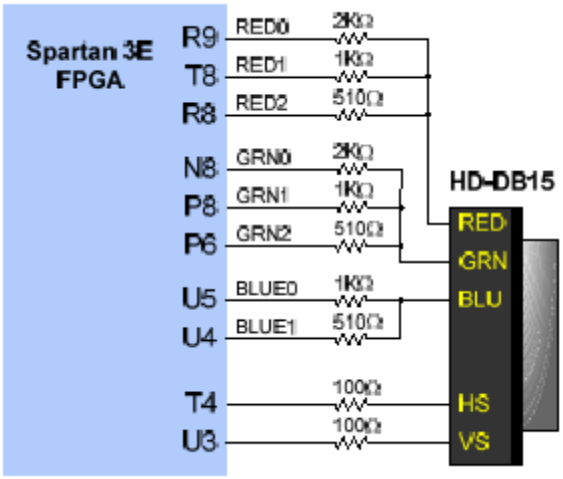


Figure 3.9 : La connexion du port VGA et le circuit FPGA.

Nous observons dans la figure 3.9 la présence d'un convertisseur numérique-analogique très simple basé sur un réseau de résistances (510Ω≈500Ω, 1k Ω =2x500Ω, 2kΩ =4x500Ω) et la sommation de courant. Nous aurons ainsi 8 niveaux de tension pour le rouge, 8 pour le vert et 4 pour le bleu (8x8x4=256) ce qui fait 256 en tout. La résistance de 500Ω est choisie en rapport avec l'impédance d'entrée de l'interface VGA pour avoir un niveau de tension entre 0,7v et 1,0v définie par la norme.

3.3.3 La conception fonctionnelle :

Le but de cette conception fonctionnelle est la familiarisation avec les outils de compilation et de simulation en VHDL ainsi que les outils de synthèse et aborder les aspects du langage VHDL suivants :

- ✓ Description structurelle et les niveaux hiérarchiques.
- ✓ L'utilisation des paquetages standards.
- ✓ Les paramètres génériques pour les temps de calcul et les tailles des composants.
- ✓ L'utilisation des composants internes ROM comme des constantes (ces composants figurent dans la deuxième conception du circuit).

Dans un premier temps, on va détailler une architecture matérielle primaire qui était ma première proposition. Elle a été ensuite décomposée en plusieurs blocs simples. Ces derniers ont été synthétisés afin de réaliser notre application.

Dans un deuxième temps, on a pu concevoir une autre architecture plus complexe en utilisant des blocs de ROM internes pour améliorer l'affichage. Enfin, on ajustera les résultats des deux conceptions pour mettre une comparaison entre eux dans différents points (temps, surface,…).

Dans les deux conceptions, la configuration du composant FPGA se fait par une liaison USB2, le résultat est recueilli sur le connecteur VGA, le jeu est animé par l'utilisation des boutons poussoirs (*Push Buttons*) (Figure 3.10).

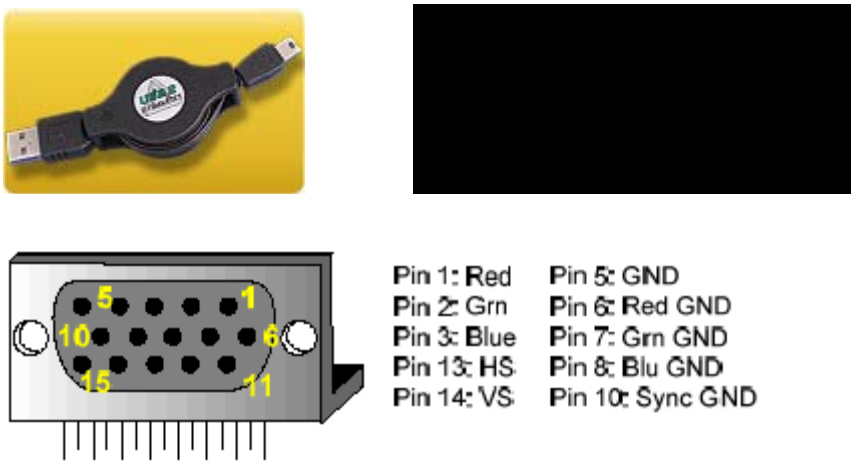


Figure 3.10 : Les éléments utilisés de la carte NEXYS2.

3.3.3.1 La première conception matérielle :

Cette architecture possède 6 blocs (Figure 3.11) :

- Le bloc DIV_N (diviseur de fréquence par N) pour adapter les horloges transmis aux blocs suivants.
- Le bloc SYNC_VGA pour générer les signaux de synchronisation.
- Un bloc RAQ_XY pour déterminer les coordonnées (a, b) du motif raquette.
- Un bloc BAL_XY pour déterminer les coordonnées (c, d) de la balle.
- Un bloc AFF_RAQ pour localiser la zone du motif raquette.
- Un bloc AFF_BAL pour localiser la zone du motif balle.
- Le dernier bloc SIG_RGB qui affecte aux zones des objets localisés les couleurs.

Les descriptions en langage VHDL de chaque bloc se trouve sur le CDROM.

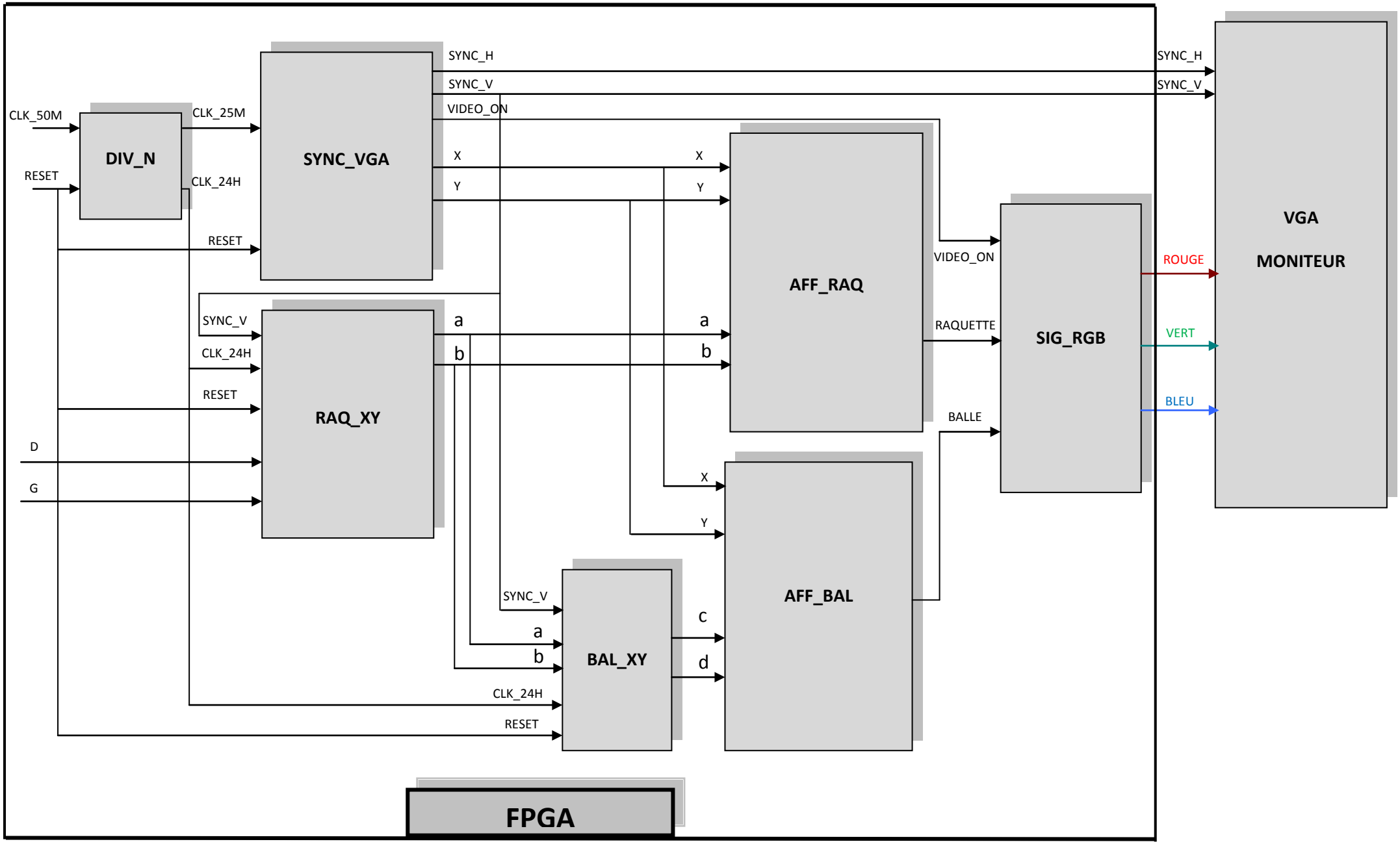


Figure 3.11 : La première conception matérielle de notre projet.

Détail des blocs :

3.3.3.1.1 Le diviseur de fréquence « DIV_N » :

L'horloge principale de la carte de développement (NEXYS2) est de 50MHz, et pour le bon fonctionnement des blocs (SYNC_VGA, RAQ_XY), nous avons besoin d'un diviseur de fréquence qui a pour fonction de ralentir les signaux envoyés à l'écran VGA et aux compteurs utilisés dans le bloc RAQ-XY. Pour cela, nous avons choisi de réaliser un bloc DIV_N, ce dernier utilise N blocs diviseur de fréquence par 2 (DIV_2) en cascades. An final nous avons deux fréquences d'horloge, une de 25MHz nécessaire pour les signaux VGA et l'autre de quelques dizaines de Hz pour l'animation du jeu.

A/ Le diviseur de fréquence « DIV_2 » :

Description fonctionnelle :
En VGA standard, l'écran doit être piloté à 25MHz (cf. § 3.2.3) et comme l'horloge principale de notre carte NEXY2 est de 50 MHz, nous avons utilisé un bloc DIV_2 qui a pour fonction de ralentir les signaux envoyés à l'écran. Ce bloc est un diviseur de fréquence par 2.
Le bloc DIV_2 (Figure 3.12) possède deux signaux d'entrées :

- CLK est l'horloge principale de la carte NEXYS2, elle est de 50MHz.
- Le RESET : on utilise un bouton poussoir pour la remise à zéro de tous les systèmes séquentiels utilisés dans notre application.

On trouve une horloge d'une fréquence de 25MHz (CLK_25MHz) comme un signal de sortie de ce bloc.

Vue externe du bloc :

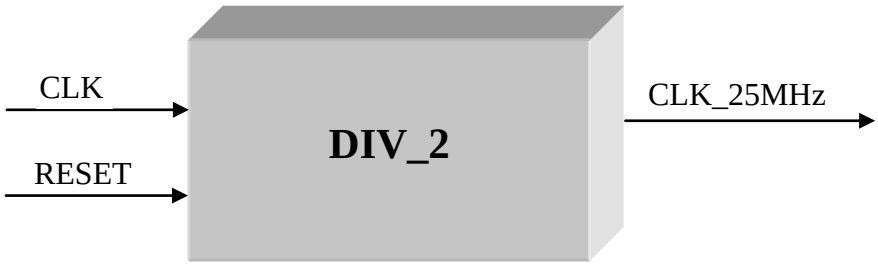


Figure 3.12 : Vue externe du bloc DIV_2.

Dans ISE 9.2i notre bloc est représenté en mode schématique par la figure 3.13 :

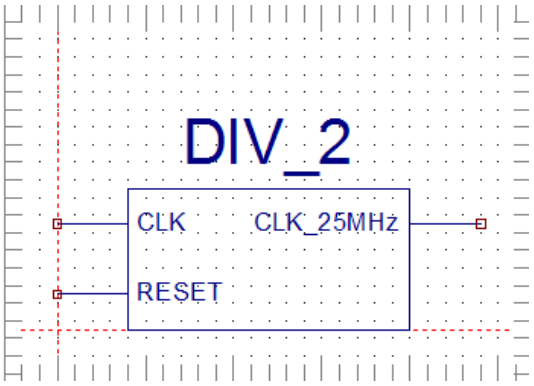


Figure 3.13 : Le symbole schématique du bloc DIV_2 à partir du logiciel.

Structure interne :

Le diviseur de fréquence est un système électronique qui permet d'obtenir un signal périodique de fréquence F2, à partir d'un signal périodique de fréquence F1. On peut réaliser le diviseur de fréquence par deux à partir d'une bascule D dont la sortie inversée est connectée à son entrée, cela nous a conduits à une description comportementale d'un circuit logique en utilisant le langage VHDL en exemple :

```
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
architecture DIV_2 of DIV_2 is
signal tmp : std_logic;
begin
--------------------------------------------------------------------------------------------------------------
CLK_25MHz <= tmp;
  process(CLK,RESET)
begin
 if RESET ='1' then tmp <= '0';
  else if CLK='1'and CLK'event then     --detection d'un front montant de l' horloge
             tmp <= not tmp;

             end if;
       end if;
end process;
end DIV_2;
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
```

Description RTL :

Après la synthèse de notre bloc, on peut avoir la description RTL du circuit sur la figure ci-dessous (Figure 3.14), on peut dire que notre bloc est synthétisable donc réalisable.

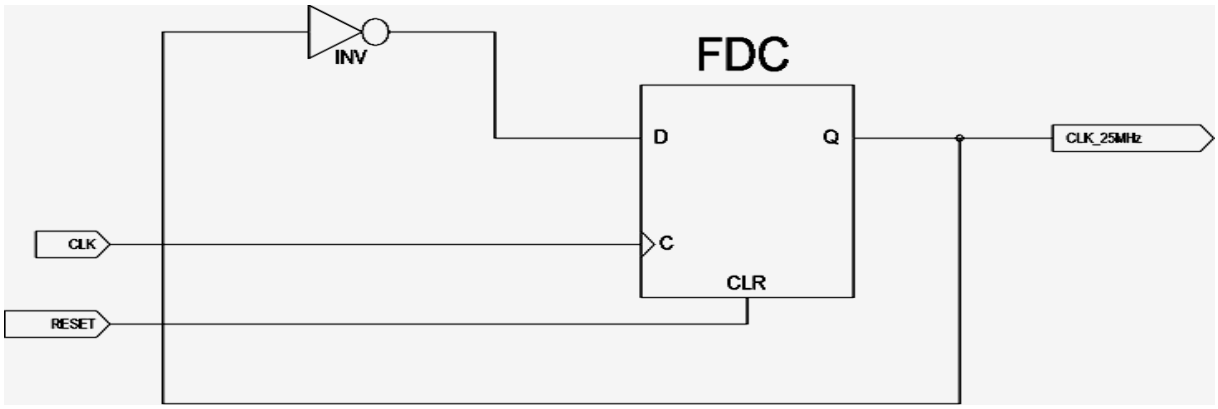


Figure 3.14 : La description RTL du bloc DIV_2.

B/ Le diviseur de fréquence « DIV_N » :

Description fonctionnelle :

On a besoin aussi de ralentir les signaux envoyés aux compteurs utilisés dans le bloc RAQ_XY, afin d'avoir une visualisation lisible du déplacement à gauche et à droite du motif de la raquette à l'aide des boutons poussoirs (il faut que le compteur soit cadencé par une horloge de quelques dizaines de Hz). Pour cela, nous avons mis le bloc DIV_2 « N » fois en cascade pour avoir un diviseur de fréquence par 2^N.

Le code VHDL de ce bloc est rédigé avec le paramètre générique N pour l'utiliser dans autres conceptions. Mais il faut définir ce paramètre dans le programme principal de l'application. Dans notre cas, nous avons choisis N = 20, ce qui donne une fréquence de ≈ 24 Hz.

Le bloc DIV_N a pour signaux d'entrées (Figure 3.15) :

- Le CLK.
- Le RESET.

Les signaux de sorties :

- CLK_25MHz : une horloge de 25MHz pour le fonctionnement de l'écran VGA.
- CLK_24Hz : une horloge de 24Hz pour le déplacement de la raquette.

Vue externe du bloc :

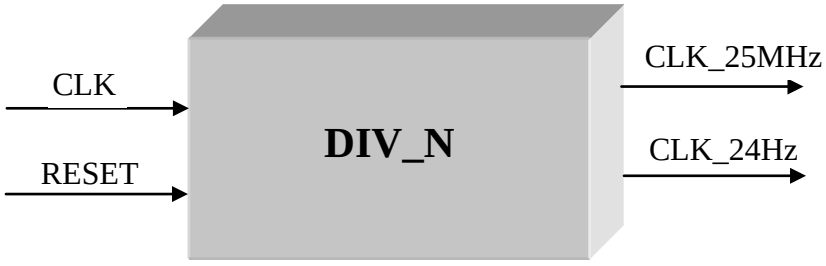


Figure 3.15 : Vue externe du bloc DIV_N.

Structure interne :

Sa structure est composée de plusieurs blocs de diviseur de fréquence par 2 (20 bascules) disposés en cascade afin d'obtenir une fréquence d'horloge mieux adapter au fonctionnement des compteurs utilisés dans le bloc RAQ_XY.

3.3.3.1.2 Le contrôleur d'écran « SYNC_VGA » :

Description fonctionnelle :
L'affichage sur un écran VGA est réalisé à l'aide d'un spot lumineux qui traverse un écran (cf. § 3.2.3). Il est nécessaire d'avoir deux signaux de synchronisation pour le bon fonctionnement d'un écran VGA d'une résolution de 640x480.
On trouve dans ce bloc (Figure 3.16) comme signaux d'entrées :

- Le CLK_25MHz.
- Le RESET.

Comme signaux de sorties :

- SYNC_H : le signal de synchronisation horizontale.
- SYNC_V : le signal de synchronisation verticale.
- Le signal VIDEO_ON qui indique la zone où l'affichage est autorisé.
- Les deux vecteurs X, Y : les coordonnées du pixel actif sur l'écran.

Vue externe du bloc :

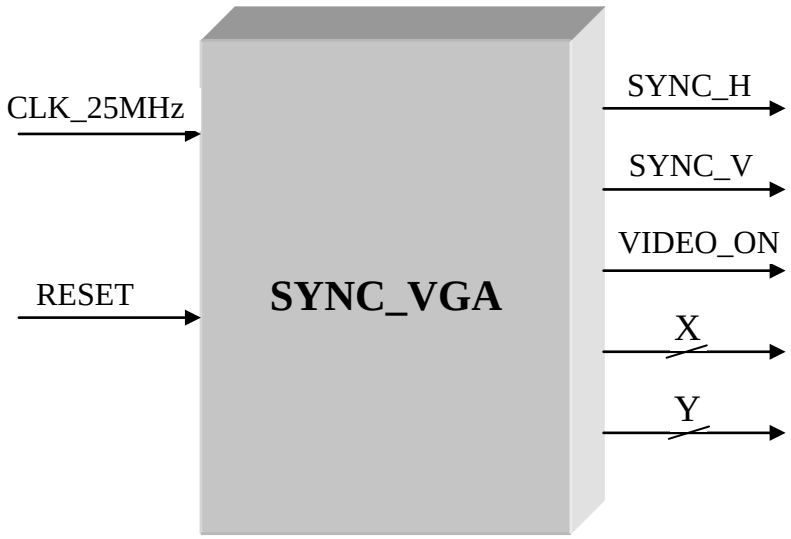


Figure 3.16 : Vue externe du bloc SYNC_VGA.

Structure interne :

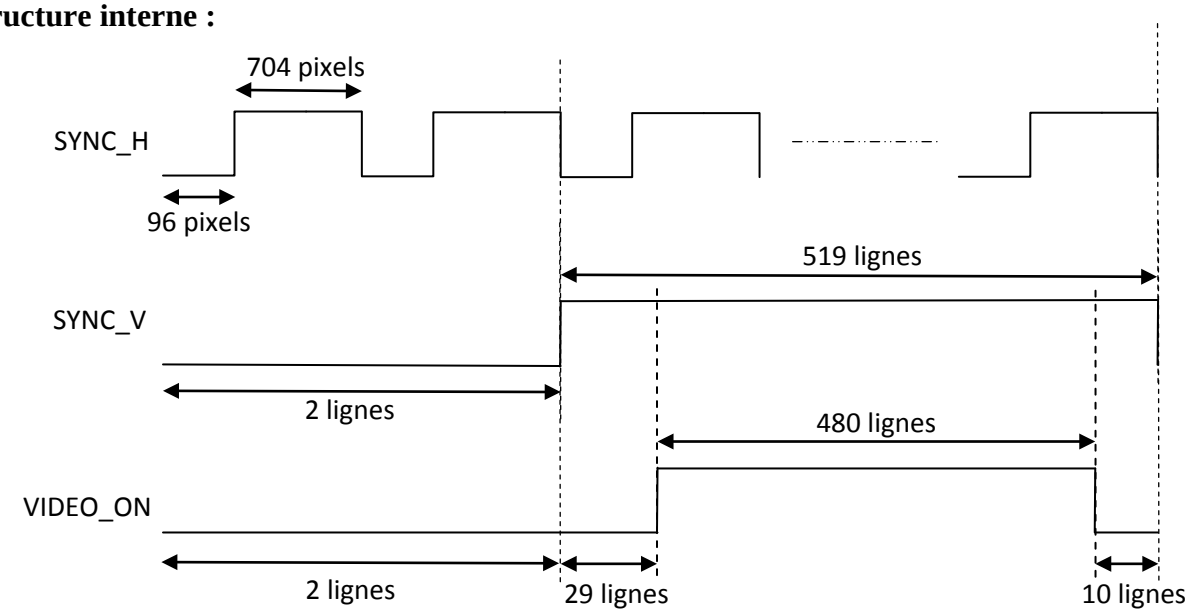


Figure 3.17 : Chronogramme des signaux de synchronisation horizontale et verticale.

Dans ce bloc, on utilise deux compteurs : Un compteur de pixel (hs_compt) et un compteur de ligne (vs_compt) qui sont nécessaires pour faire le balayage de l'écran et réaliser le timing de la trame VGA (Figure 3.17). Le compteur ligne est incrémenté à chaque fois que le compteur pixel termine 800 pixels. Nous avons utilisé des comparateurs pour générer les signaux de synchronisation horizontale et verticale (SYNH_H, SYNH_V) et localiser la zone autorisée pour l'affichage (VIDEO_ON). Des soustracteurs ont été utilisés pour trouver les coordonnées du pixel actif (X, Y) (Figure 3.18). Les coordonnées (X, Y) repère le pixel actif à l'intérieur de la zone d'affichage, (X=0, Y=0) pour le coin haut à gauche et (X=639, Y=479) pour le coin bas à droite.

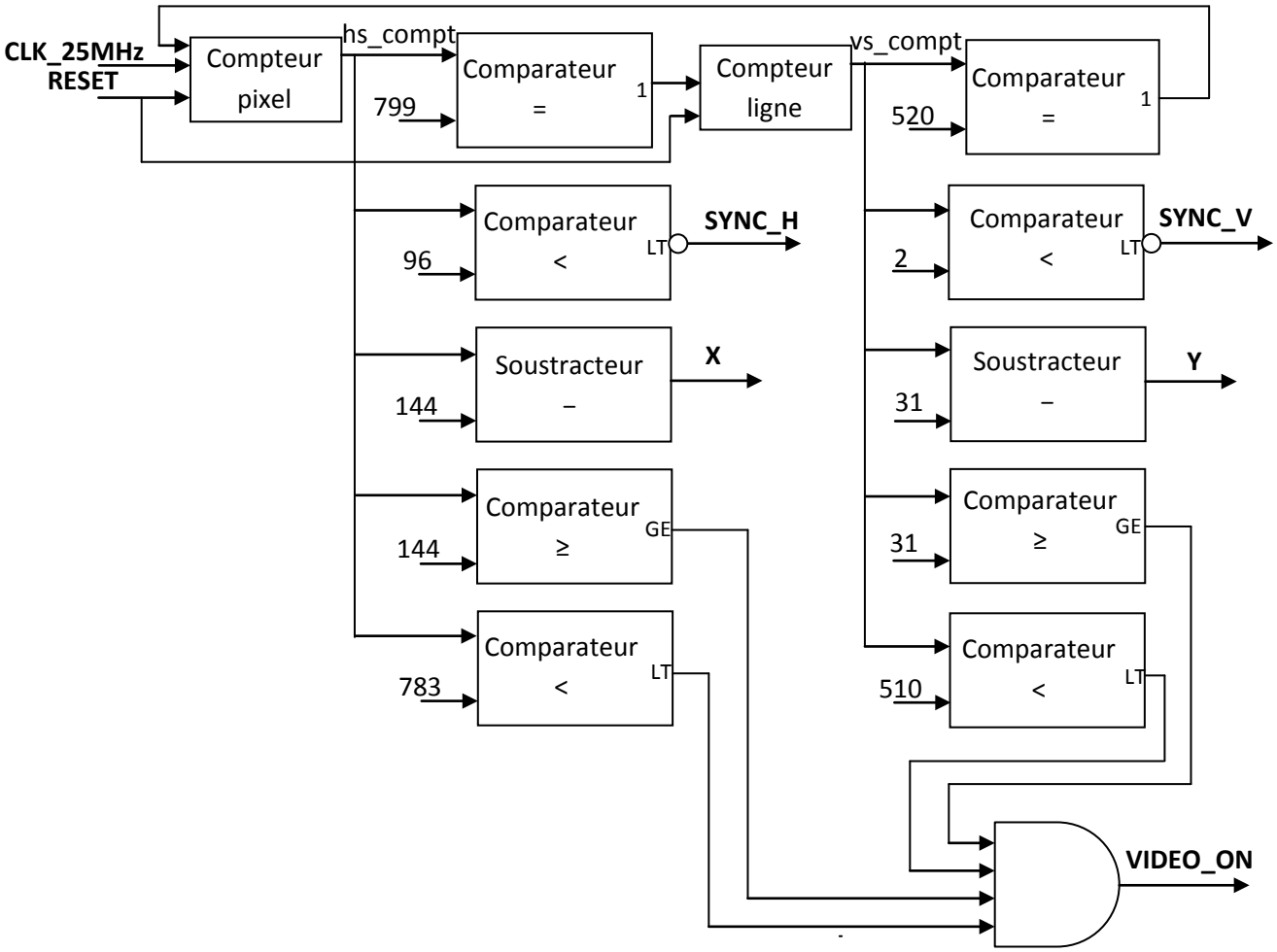


Figure 3.18 : Vue interne du bloc SYNC_VGA.

3.3.3.1.3 Les coordonnées de la raquette « RAQ_XY » :

Description fonctionnelle :

Ce bloc permet de positionner le motif de la raquette en calculant les coordonnées du point extrême haut et gauche du motif. A l'aide de deux boutons poussoirs, on peut déplacer la raquette vers la droite en appuyant sur le bouton D et vers la gauche à l'aide du bouton G. La raquette se déplace sur le bord bas de l'écran.

Les signaux d'entrées de ce bloc (Figure 3.19) sont :

- CLK_24Hz : fréquence d'horloge adaptée pour une bonne visualisation du déplacement du motif raquette.
- Le RESET.
- Le SYNC_V qui a pour rôle de réactualiser ces coordonnées à la fréquence image.
- D : le niveau actif de ce signal permet le déplacement à droite de la raquette.
- G : le niveau actif de ce signal permet le déplacement à gauche de la raquette.

Les signaux de sorties sont les coordonnées (a, b) du point extrême (haut et gauche) de la raquette.

Vue externe du bloc :

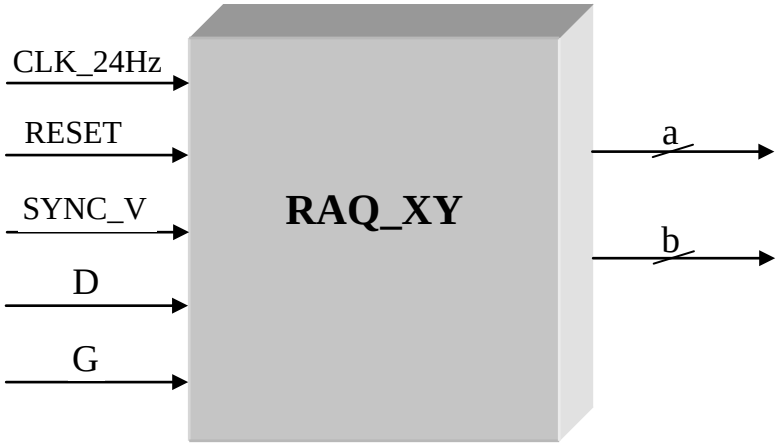


Figure 3.19 : Vue externe du bloc RAQ_XY.

Structure interne :

Dans ce bloc (Figure 3.20), on utilise un compteur/décompteur (Raquet_X) qui est nécessaire pour le déplacement de la raquette. Le compteur sera incrémenté à l'état actif du signal D avec un pas d'une valeur V (V est en rapport avec la rapidité du déplacement, nous avons fait des essais avec V=1), puis il sera comparé avec le bord droit de l'écran (Max = 639 – Lr - V) afin de s'arrêter. A l'état actif du signal G, le compteur (Raquet_X) sera décrémenté pas à pas jusqu'à ce qu'il arrive au bord gauche de l'écran en le comparant avec la valeur Min (Min = V). La coordonnée 'a' sera la valeur du compteur Raquet_X. La coordonnée 'b' sera fixe (b = 479 - Hr) car la raquette ne se déplace pas sur l'axe Y.

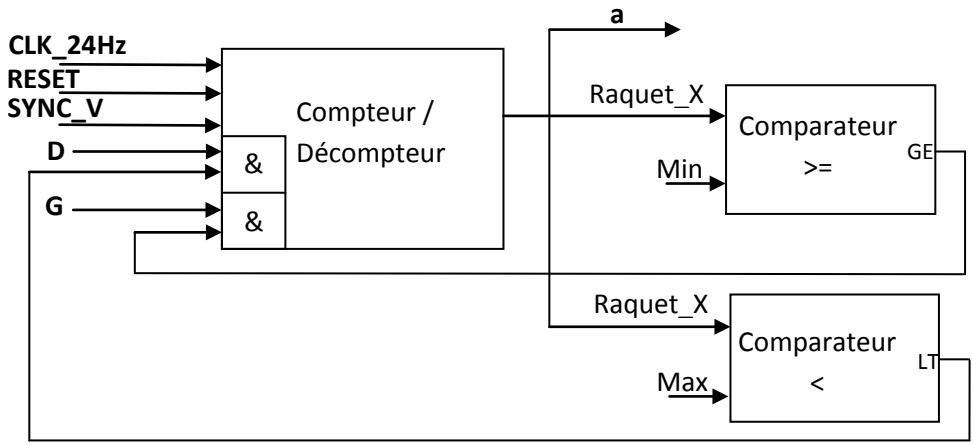


Figure 3.20 : Vue interne du bloc RAQ_XY.

3.3.3.1.4 Les coordonnées de la balle « BAL_XY » :

Description fonctionnelle :

Ce bloc a pour rôle de définir la trajectoire de la balle, en calculant les différents emplacements du motif. En touchant l'un des trois bords de l'écran (haut, gauche et droit) ou la raquette, la balle rebondisse et change sa trajectoire mais lorsqu'elle touche le bord bas, elle doit s'arrêter et la partie est finie.

Les signaux d'entrées de ce bloc (Figure 3.21) sont :

- CLK_24Hz.
- Le RESET.
- Le SYNC_V.
- Les coordonnées (a, b) de la raquette (calculées dans le bloc précédent).

Les signaux de sorties sont les coordonnées (c, d) du point extrême (haut et gauche) de la balle.

Vue externe du bloc :

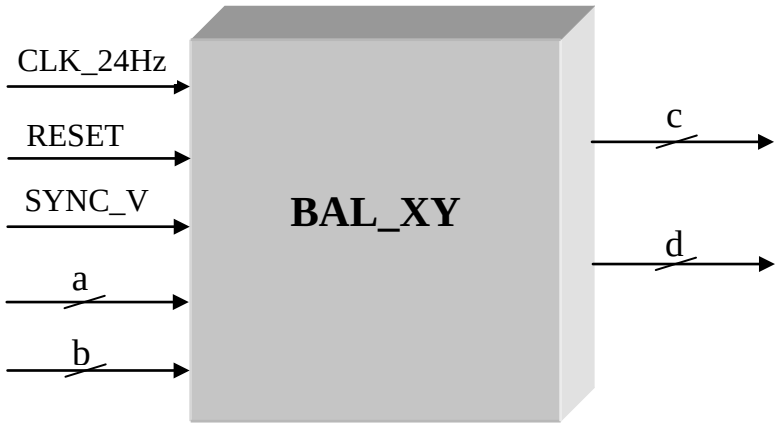


Figure 3.21 : vue externe du bloc BAL_XY.

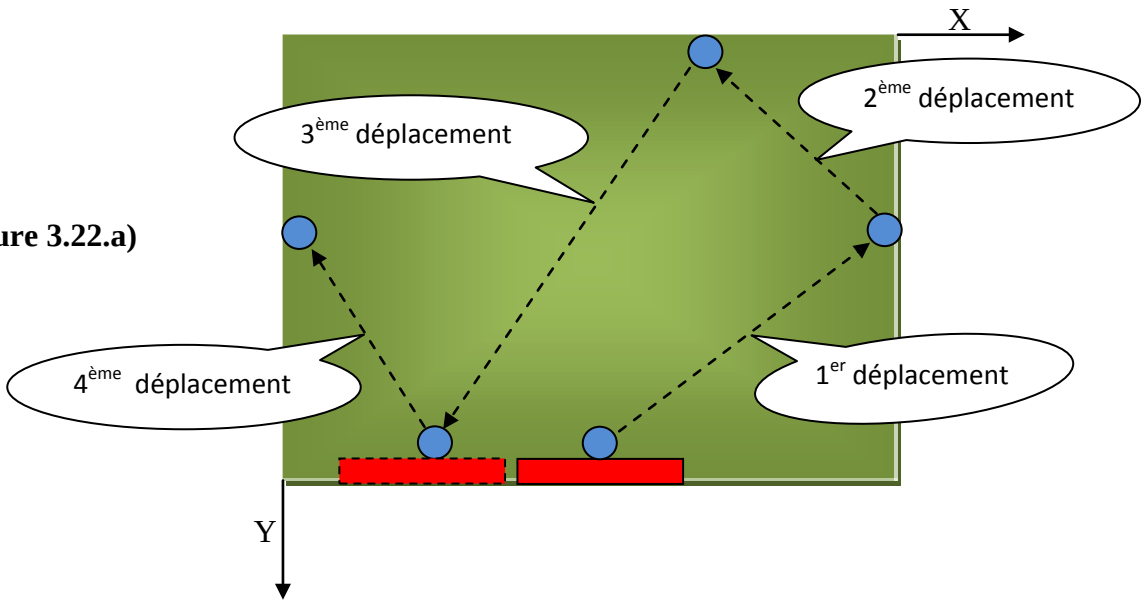
Structure interne :

Pour déterminer les coordonnées (c, d) de la balle, on utilise deux compteurs/décompteurs (Ball_X, Ball_Y) qui s'incrémentent ou se décrémentent selon la position de la balle (Ball_X : compteur suivant l'axe des X. Ball_Y : compteur suivant l'axe des Y). Et pour localiser la position de la balle, on a utilisé des comparateurs.

La trajectoire de la balle est résumé dans la figure 3.22 selon deux situations (Figure 3.22.a) et (Figure 3.22.b) :

- Au premier déplacement : le compteur Ball_X s'incrémente et le compteur Ball_Y se décrémente jusqu'à ce que la balle arrive au bord droit de l'écran (Max_X = 639 – Lb).
- Au 2ème déplacement : Ball_X se décrémente, Ball_Y se décrémente jusqu'à ce que la balle arrive au bord haut de l'écran (Min_Y = 0).
- Au 3ème déplacement : Ball_X se décrémente, Ball_Y s'incrémente jusqu'à ce que la balle touche la raquette (Figure 3.22.a) et passe au 4ème déplacement, ou elle manque la raquette (Figure 3.22.b) et les deux compteurs (Ball_X, Ball_Y) se bloquent, la balle s'arrête et la partie est finie.
- Au 4ème déplacement (Figure 3.22.b) : Ball_X se décrémente, Ball_Y se décrémente jusqu'à ce que la balle touche le bord gauche de l'écran (Min_X = 0).

(Figure 3.22.a)



(Figure 3.22.b)

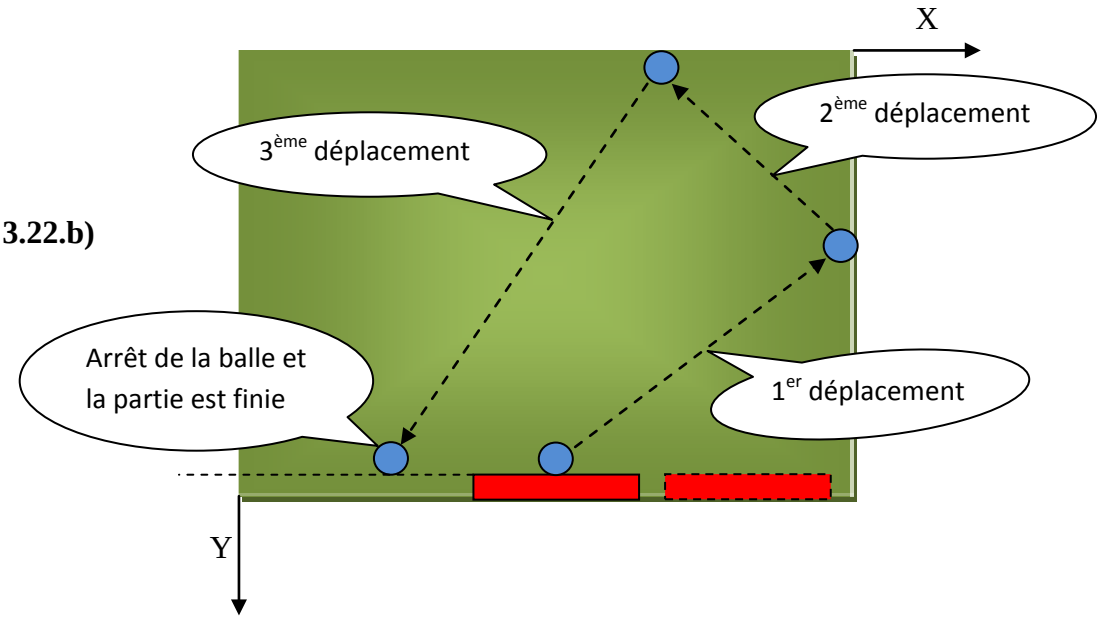


Figure 3.22 : La trajectoire de la balle.

On peut constater la structure interne de notre bloc sur la figure 3.23 :

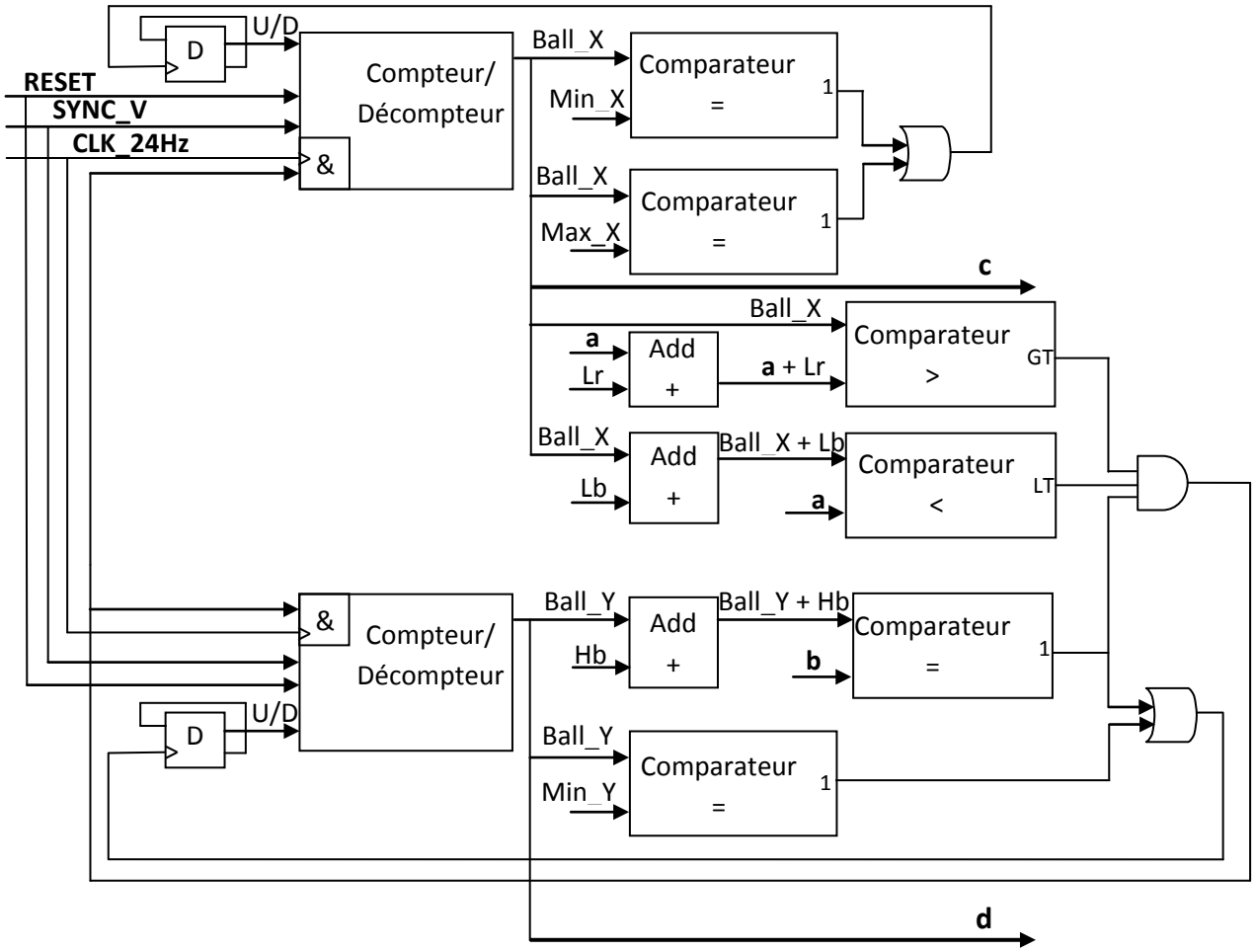


Figure 3.23 : Vue interne du bloc BAL_XY.

3.3.3.1.5 Localisation du motif raquette « AFF_RAQ » :

Description fonctionnelle :

Ce bloc permet de localiser la zone du motif raquette dans l'écran, en tenant compte des coordonnées (a, b), de la longueur Lr et de la hauteur Hr de la raquette.

Les signaux d'entrées de ce bloc (Figure 3.24) sont :

- Les coordonnées du pixel actif (X, Y).
- Les coordonnées (a, b) de la raquette.

RAQUETTE étant son signal de sortie qui indique la zone des pixels qui appartiennent au motif raquette.

Vue externe du bloc :

Figure 3.24 : Vue externe du bloc AFF_RAQ.

Structure interne :

Pour afficher l'objet raquette (Figure 3.25), il faut comparer la position du pixel actif de coordonnées (X, Y) et les coordonnées (a, b) de la raquette tous en ajoutant la longueur (Lr) et la hauteur (Hr) du motif de la raquette, ceci est réalisé matériellement par un ensemble de comparateur tel que décrit par le schéma RTL de la figure 3.26.

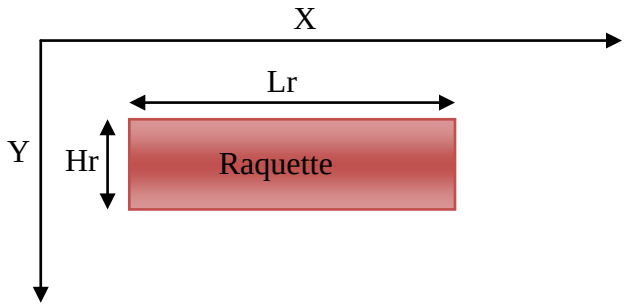


Figure 3.25 : La position de la raquette par rapport aux coordonnées (X, Y).

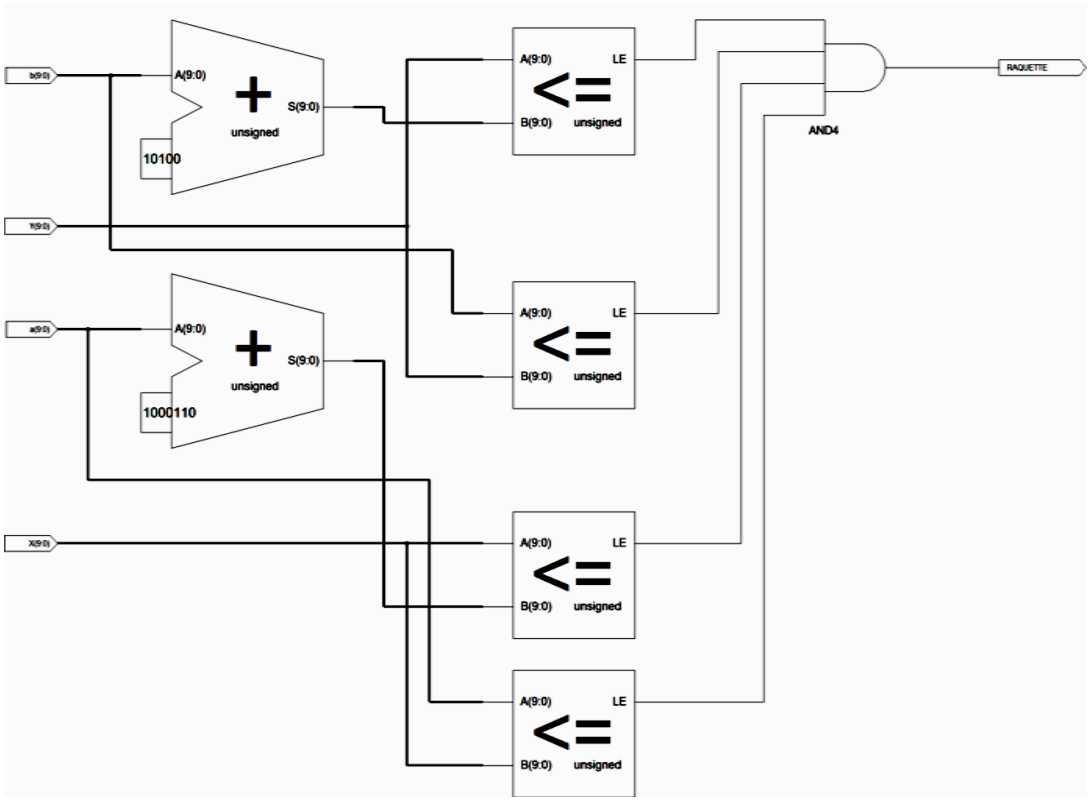


Figure 3.26 : La description RTL du bloc AFF_RAQ.

3.3.3.1.6 Localisation du motif balle « AFF_BAL» :

Description fonctionnelle :

Ce bloc permet de limiter la zone du motif balle dans l'écran, en tenant compte des coordonnées (c, d), de la longueur Lb et de la hauteur Hb de la balle.

Les signaux d'entrées de ce bloc (Figure 3.27) sont :

- Les coordonnées du pixel (X, Y).
- Les coordonnées (c, d).

BALLE étant son signal de sortie qui indique la zone des pixels qui appartiennent au motif balle.

Vue externe du bloc :

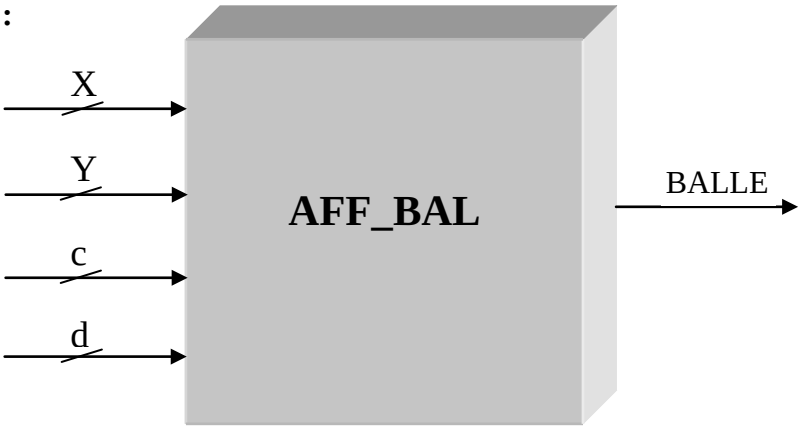


Figure 3.27 : Vue externe du bloc AFF_BAL.

Structure interne :

Pour afficher l'objet balle (Figure 3.28), il faut comparer la position du pixel actif de coordonnées (X, Y) et les coordonnées (c, d) de la balle tous en ajoutant la longueur (Lb) et la hauteur (Hb) du motif de la balle, ceci est réalisé matériellement par un ensemble de comparateur tel que décrit par le schéma RTL de la figure 3.29

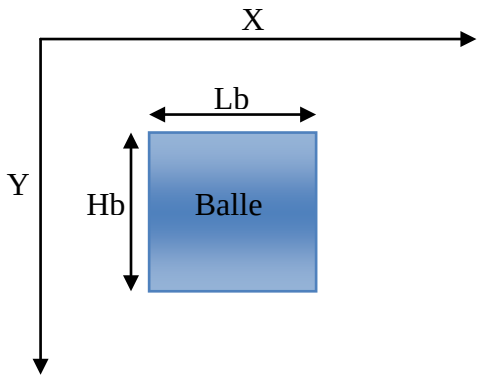


Figure 3.28 : La position de la balle par rapport aux coordonnées (X, Y).

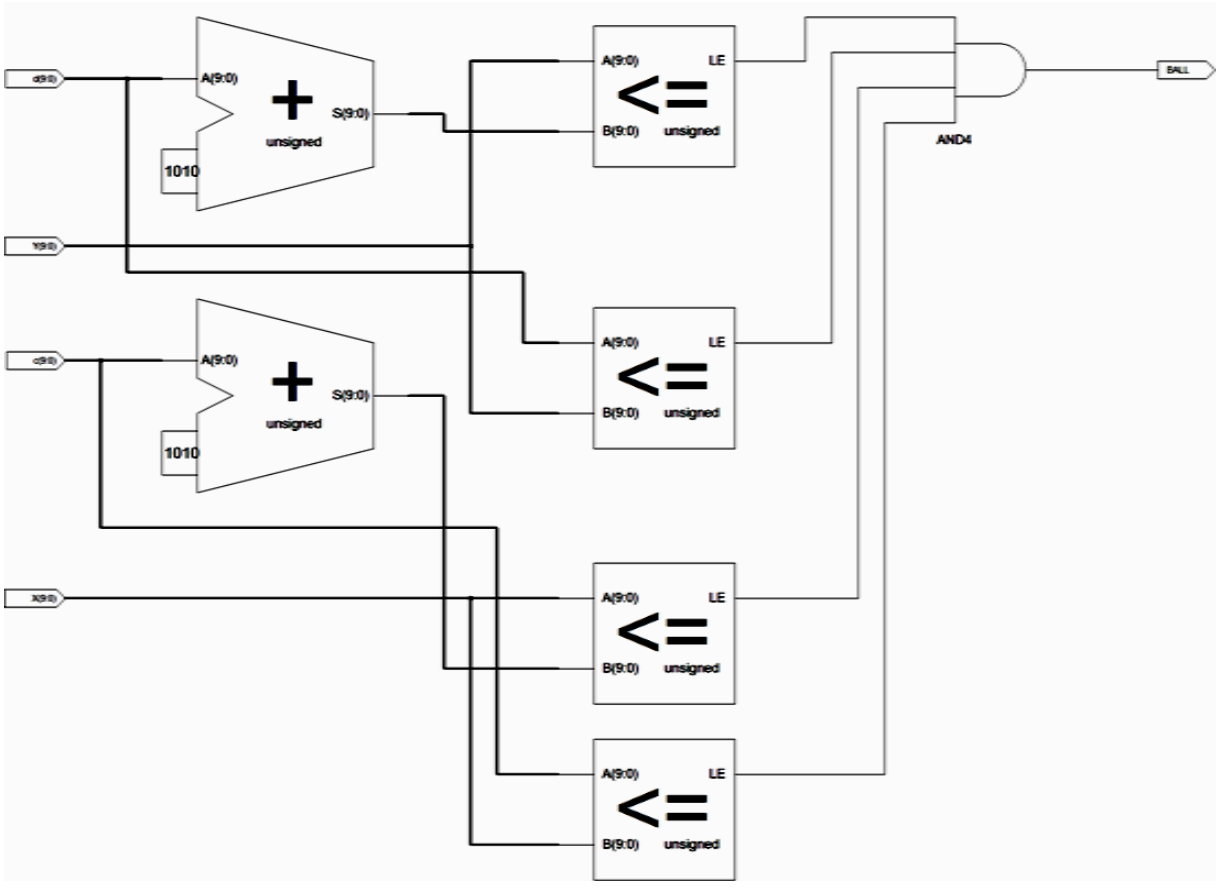


Figure 3.29 : La description RTL du bloc AFF_BAL.

3.3.3.1.7 Le signal RVB « SIG_RGB » :

Description fonctionnelle :

Ce bloc permet d'associer aux zones localisées des objets les couleurs RVB (*RGB red green bleu*).
Les signaux d'entrées de ce bloc (Figure 3.30) sont :

- Le signal VIDEO_ON qui autorise le moment d'affichage.
- RAQUETTE.
- BALLE.

Les signaux de sorties sont les couleurs primaires qui sont affectées à tous pixels de l'écran.

Vue externe du bloc :

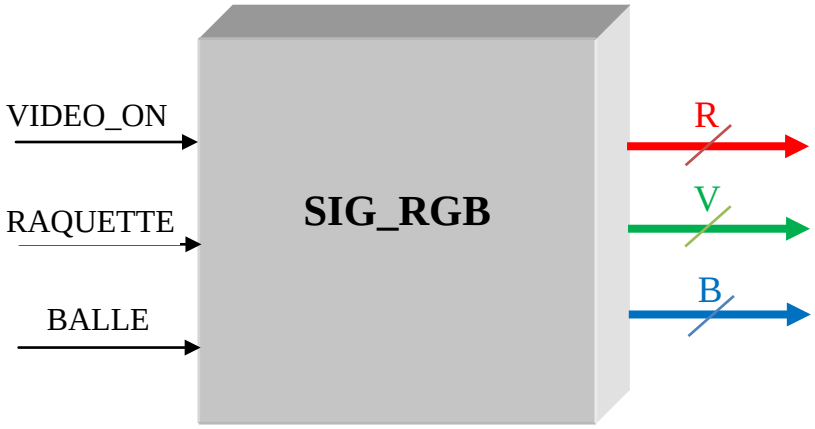


Figure 3.30 : Vue externe du bloc SIG_RGB.

Structure interne :
On peut définir le principe de fonctionnement de ce bloc selon le tableau 3.2.

VIDEO_ON	RAQUETTE	BALLE	R	V	B	Région
0	X	X	0	0	0	Extinction du pixel actif
1	1	0	1	0	0	La raquette en rouge
1	0	1	0	1	0	La balle en bleu
1	0	0	0	0	1	Le fond en vert

Tableau 3.2 : Le principe de fonctionnement du bloc SIG_RGB.

A l'état inactif du signal VIDEO_ON, on éteint le pixel actif de l'écran (pas d'affichage).
A l'état actif du signal VIDEO_ON, on distingue trois situations :

- A l'état actif du signal RAQUETTE, RVB = 100 ceci permet d'afficher la raquette en rouge.
- A l'état actif du signal BALLE, RVB = 001 ceci permet d'afficher la balle en bleu.
- A l'état inactif des deux signaux précédents, RVB = 010 ceci permet d'affecter la couleur verte au fond.

On peut présenter la structure interne de ce bloc sur la figure 3.31 :

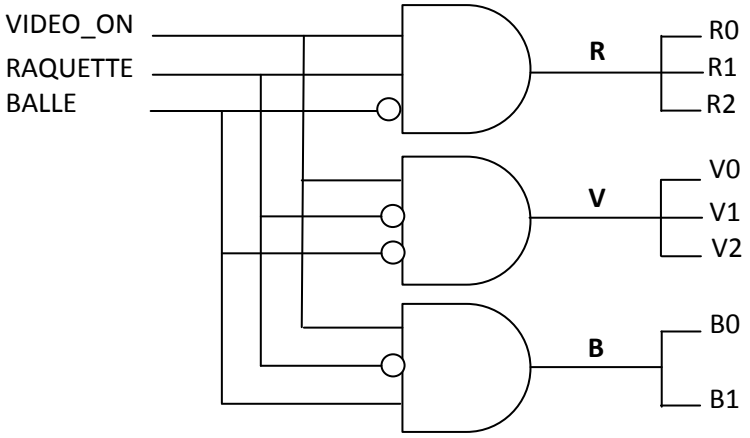


Figure 3.31 : Vue interne du bloc SIG_RGB.

3.3.3.2 La deuxième conception matérielle :

Nous avons opté pour la conception d'une deuxième architecture afin d'obtenir des meilleurs performances en terme de forme des motifs. Dans la première architecture, nous n'avons pas pu réaliser la forme ronde de la balle avec des blocs simples. Pour cela, nous avons choisi de procéder avec des blocs de ROM afin de mémoriser les formes, les couleurs ainsi que les tailles des motifs utilisés dans le jeu vidéo.

Cette architecture possède 9 blocs (Figure 3.32), les 4 premiers blocs sont les mêmes blocs de la première conception :

- Le bloc DIV_N (diviseur de fréquence par N). (cf. § 3.3.3.1.1)
- Le bloc SYNC_VGA. (cf. § 3.3.3.1.2)
- Le bloc RAQ_XY. (cf. § 3.3.3.1.3)
- Le bloc BAL_XY. (cf. § 3.3.3.1.4)
- DEC_ADRESS pour adresser à une mémoire parmi plusieurs.
- Un bloc ROM_1 : contient le motif de la raquette.
- Un bloc ROM_2 : contient le motif de la balle.
- Un bloc ROM_3 : contient la couleur des pixels qui appartient au fond du jeu.
- Un multiplexeur MUX qui permet d'afficher l'image

Les descriptions en langage VHDL des nouveaux blocs se trouve sur CDROM.

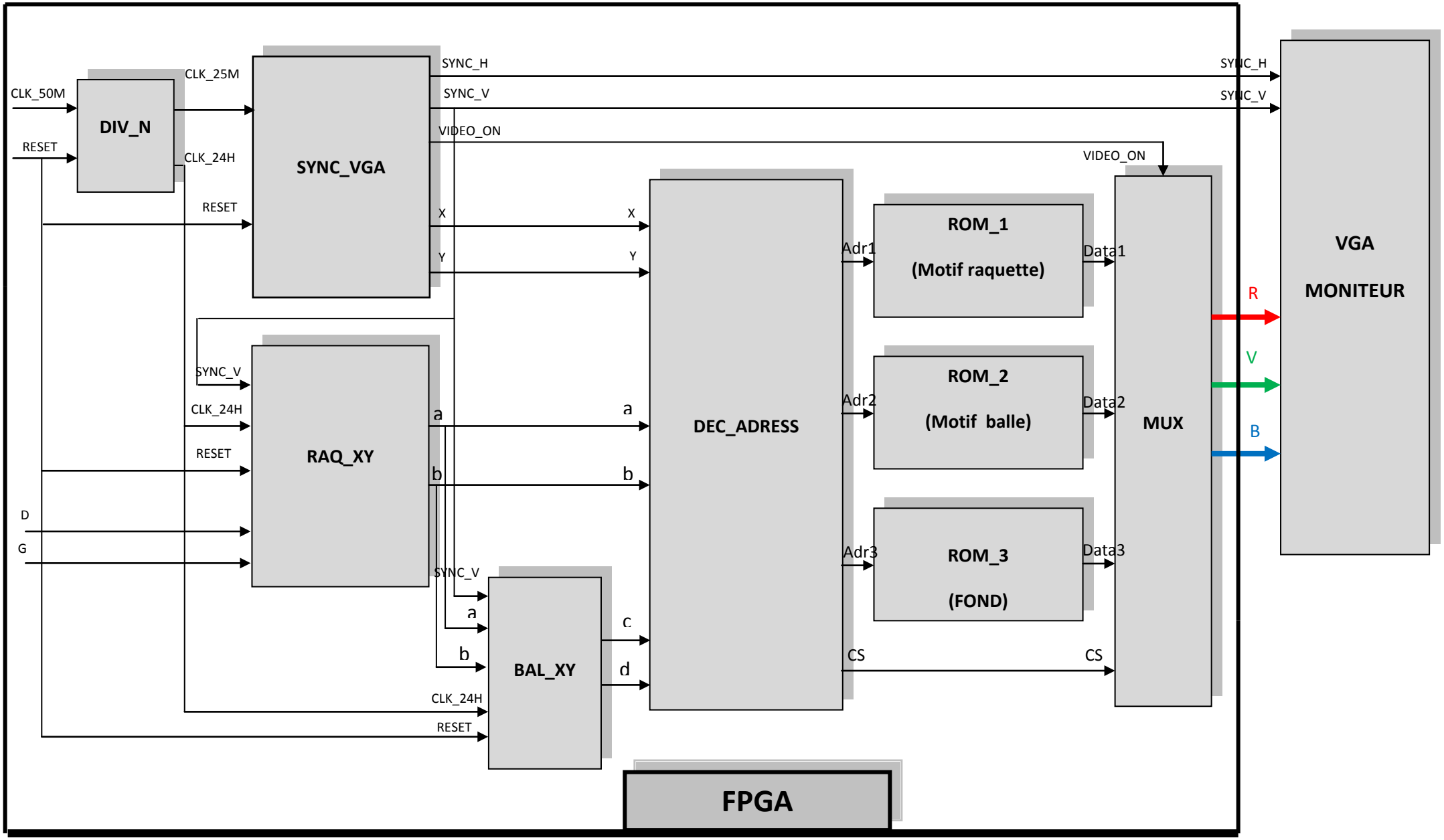


Figure 3.32 : La deuxième conception matérielle de notre projet.

Détail des nouveaux blocs :

3.3.3.2.1 Le décodeur d'adresse « DEC_ADRESS » :

Description fonctionnelle :
Ce bloc permet de générer les adresses pour les mémoires ROM de chaque objet tout en respectant l'espace de ce dernier (Figure 3.33).
Les signaux d'entrées de ce bloc sont :

- Les coordonnées du pixel actif (X, Y) de l'écran.
- Les coordonnées (a, b) de la raquette.
- Les coordonnées (c, d) de la balle.

Les signaux de sorties sont :

- Les adresses (Adr1, Adr2 et Adr3) des trois ROM (ROM_1, ROM_2 et ROM_3) respectivement.
- Le CS (*chip select*) pour permettre la sélection d'une ROM parmi les trois à l'aide d'un multiplexeur.

Vue externe du bloc :

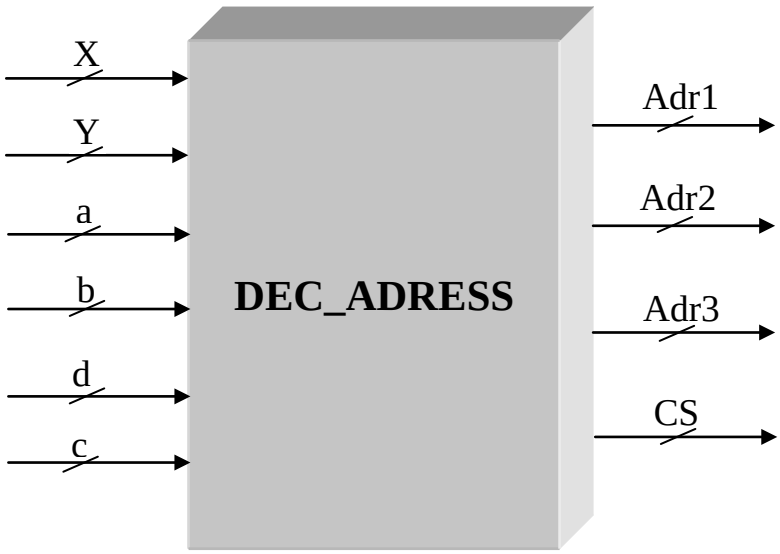


Figure 3.33 : Vue externe du bloc DEC_ADRESS.

Structure interne :
Le décodeur d'adresse utilise des comparateurs pour comparer les coordonnées (X, Y) du pixel actif aux coordonnées (a, b) de la raquette et (c, d) de la balle afin de déterminer l'objet à afficher et sélectionner sa ROM correspondante.

3.3.3.2.2 La mémoire contenant le motif de la raquette « ROM_1 » :

Description fonctionnelle :
La forme du motif de la raquette sera stockée dans la mémoire ROM_1 (Figure 3.34).Les signaux d'entrées sont les adresses provenant de décodeur d'adresse (Adr1). La sortie est la couleur du pixel à afficher (Data1), elle est codée sur un octet.

Vue externe du bloc :

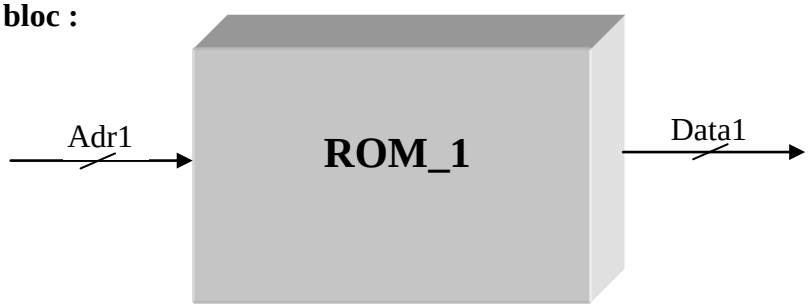


Figure 3.34 : Vue externe du bloc ROM_1.

Structure interne :

Une image est stockée en mémoire sous forme de collection de points élémentaires appelés pixels. Nous pouvons considérer une image numérique comme une page de nombres organisés en tableau ou une matrice. Chaque nombre représente les caractéristiques du pixel. La position de chaque pixel (Figure 3.35) peut être exprimée par deux coordonnées sur l'axe horizontal X et l'axe vertical Y.

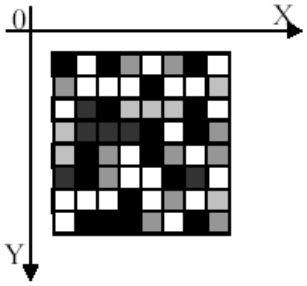


Figure 3.35 : Elément d'une image « le pixel ».

La ROM_1 contient la couleur de chaque pixel qui forme le motif de la raquette ; le pixel est codé sur un octet (RRRGGGBB) donc les données sortant du mémoire sont aussi codées sur 8 bits, précisant la couleur des pixels. Donc on peut jouer sur la forme de la raquette en définissant des couleurs précises pour chaque pixel tous en respectant la taille de la ROM qui est la taille du motif, nous avons choisi la raquette d'une longueur Lr (Lr = 60) et d'une hauteur Hr (Hr = 10):

```
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
entity rom_1 is

        Port (Adr1: in STD_LOGIC_VECTOR (9 downto 0);

              Data3: out STD_LOGIC_VECTOR (7 downto 0));

end rom_1;

--------------------------------------------------------------------------------------------------------
architecture Behavioral of rom_1 is

type memo is array (0 to 599)of STD_LOGIC_VECTOR (7 downto 0);

constant rom_1 : memo :=("00011100", --Déclaration de la rom comme constante
```

```
"00011100",
"11100000"
"11100000",
    ||      --Remplir la ROM_1 avec les couleurs
    ||      --jusqu'à 600 valeurs (taille de la rom)
"11100000",
"00011100",
```

begin

```
Data2 <= rom_2(conv_integer(Adr2)) when Adr2 <= "111111" else
"ZZZZZZZZ";
```

end Behavioral;

3.3.3.2.3 La mémoire contenant le motif de la BALLE « ROM_2 » :

Description fonctionnelle :
La forme du motif de la balle sera stockée dans une mémoire ROM_2 (Figure 3.36), les signaux d'entrées sont les adresses provenant du décodeur d'adresse (Adr2). La sortie est la couleur du pixel à afficher (Data2).

Vue externe du bloc :

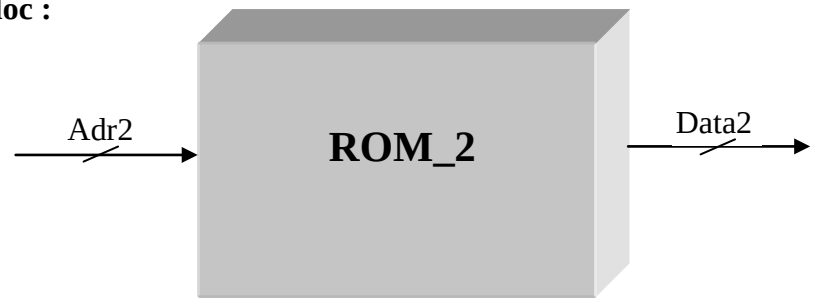


Figure 3.36 : Vue externe du bloc ROM_2.

Structure interne :
Les données de la ROM_2 est une matrice de 16x16 pixels qui définie la couleur de chaque pixel du motif de la balle. La forme de la balle doit être ronde, tout en sachant que l'objet est dessiné carré. En modifiant correctement la couleur des pixels, on obtient le motif de la balle ayant une forme arrondie. Pour cela, on défini ROM_2 comme étant une matrice constante remplie avec les différentes couleurs de chaque pixel pour obtenir une forme bien adaptée (forme ronde), on peut aussi jouer sur la couleur des pixels en créant des dessins dans la balle (Figure 3.37).

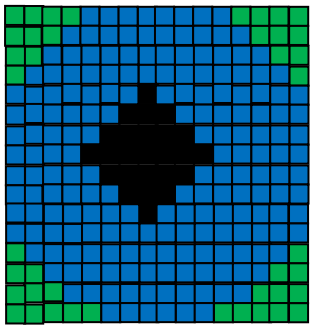


Figure 3.37 : Une matrice de 16x16 pixels.

3.3.3.2.4 La mémoire contenant la couleur des pixels du fond de l'écran « ROM_3 » :

Description fonctionnelle :
Ce bloc est aussi une mémoire de type ROM qui contient la couleur de chaque pixel du fond de l'écran de notre jeu vidéo (Figure 3.38).

Vue externe du bloc :

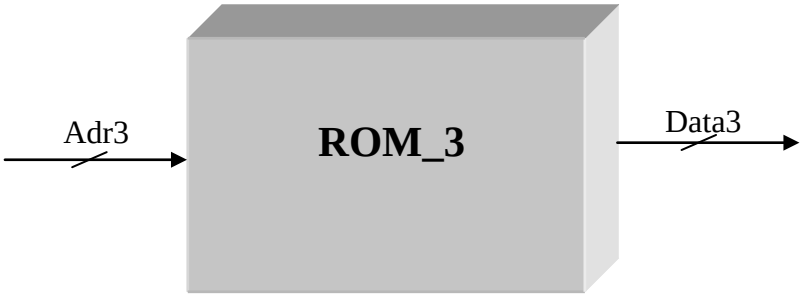


Figure 3.38 : Vue externe du bloc ROM_3.

Structure interne :
Le fond du jeu vidéo est la partie affichée de l'écran (640 x 480 pixels), donc on ne peut pas utiliser une ROM de cette taille là. Pour cela, on va créer une mémoire d'une petite taille (par exemple une matrice de 20 x 20 pixels) qui sera répétée sur toute la surface de l'écran.
Donc il faut affecter à une matrice de 400 pixels les valeurs nécessaires pour créer le motif de répétition (Dans notre cas on a choisi la même couleur verte pour le fond « 00011100 »), et pour ne pas répéter l'octet 400 fois, on a définie ROM_3 comme étant un signal dans la description comportementale (description VHDL) qui utilise un paramètre générique afin d'affecter à chaque donnée (Data3) la couleur rouge ; en exemple :

```
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
entity ROM_3 is
     Port ( Adr3 : in  STD_LOGIC_VECTOR (9 downto 0);
          Data3 : out  STD_LOGIC_VECTOR (7 downto 0));
end ROM_3;
--------------------------------------------------------------------------------------------------------
architecture ROM_3 of ROM_3 is
type memo is array (0 to 399)of STD_LOGIC_VECTOR (7 downto 0);
```

```
signal rom_3 : memo;    --Déclarer la mémoire comme étant un signal

begin

 A: for i in 0 to 399 generate  --Une boucle pour affecter à chaque pixel
rom_3(i) <= "00011100";         -- de la ROM une couleur verte

 end generate A;

 Data3 <= rom_3(conv_integer(Adr3))when Adr3 <= "110001111"
else "ZZZZZZZZ";

end ROM_3;
```

3.3.3.2.5 Le multiplexeur « MUX » :

Description fonctionnelle :

Ce bloc a pour fonction d'attribuer les données d'une ROM sélectionnée au signal final JEU du bloc RGB (Figure 3.39), permettant d'afficher chaque objet avec ses couleurs et au bon moment.

Les signaux d'entrées de ce bloc sont :

- VIDEO_ON.
- Data1 : les données de la ROM_1.
- Data2 : les données de la ROM_2.
- Data3 : les données de la ROM_3.
- CS : a pour rôle de sélectionner une des ROM.

Et on a le signal de sortie JEU qui est codé sur un octet, il présente les valeurs de RVB.

Vue externe du bloc :

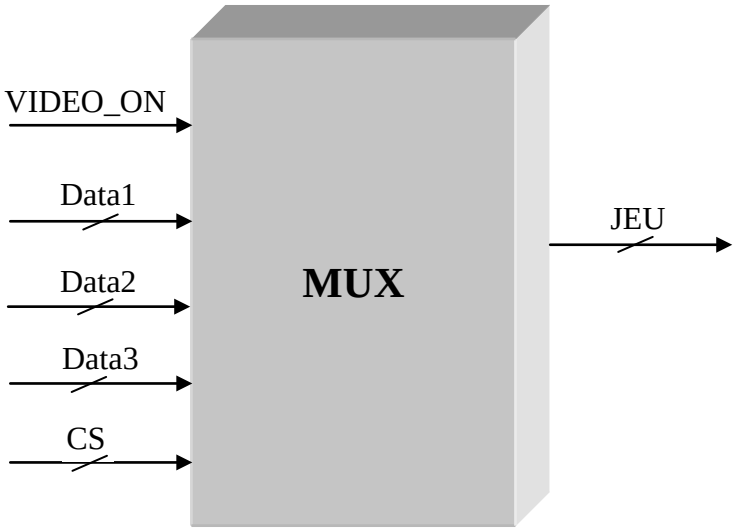


Figure 3.39 : Vue externe du bloc MUX.

Structure interne :

Au bon moment d'affichage (à l'état actif du signal VIDEO_ON), une seule ROM doit être sélectionnée à partir de la valeur de CS. Le signal JEU prend la valeur des données de la ROM sélectionnée (Figure 3.40).

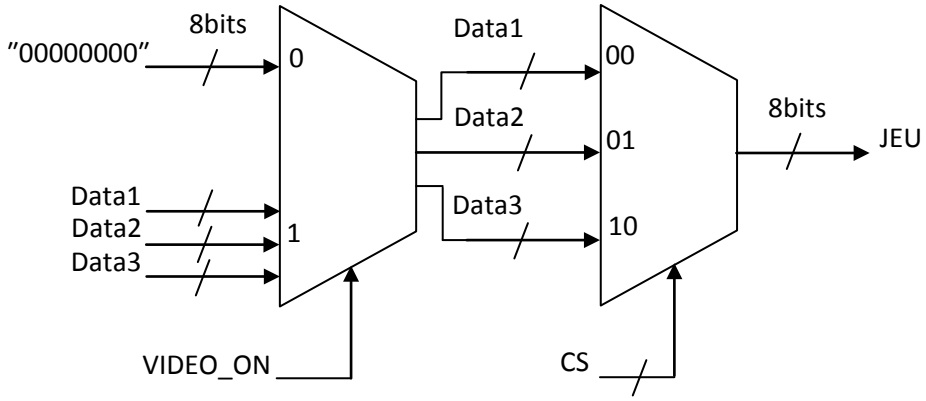


Figure 3.40 : Structure interne du bloc MUX.

3.3.4 Validation et résultats :

Après l'écriture de chaque bloc en code VHDL, il faut les simuler soit par une simulation directe, soit par une simulation avec un fichier de test (*test bench*) (cf. § 2.2.2.5). Ceci permet de vérifier le bon fonctionnement de chaque bloc et d'analyser les différents signaux d'entrées-sorties et aussi les signaux internes utilisés dans chaque bloc dans une fenêtre du logiciel dédiée à la visualisation des courbes (Figure 3.41). On a choisi comme exemple, le fichier de test du bloc : diviseur de fréquence par 2.

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
--------------------------------------------------------------------------------
ENTITY test_div_2_vhd IS
END test_div_2_vhd;
--------------------------------------------------------------------------------
ARCHITECTURE behavior OF test_div_2_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT DIV_2      --Instancier le composant DIV_2
    PORT(
            CLK : IN std_logic;
            RESET : IN std_logic;
            CLK_25MHz : OUT std_logic
            );
    END COMPONENT;
    --Inputs
    SIGNAL CLK :  std_logic := '0';
    SIGNAL RESET :  std_logic := '0';
    --Outputs
```

```
        SIGNAL CLK_25MHz :  std_logic;
BEGIN
        -- Instantiate the Unit Under Test (UUT)
        uut: DIV_2 PORT MAP(
                CLK => CLK,
                RESET => RESET,
                CLK_25MHz => CLK_25MHz);

        tb : PROCESS    --Réaliser un process pour avoir une horloge périodique
        BEGIN
CLK <= '0';wait for 100 ns;
CLK <= '1';wait for 100 ns;
  end process;
  tm : process
        begin
                -- Wait 100 ns for global reset to finish
                wait for 100 ns;
RESET <= '1';wait for 100 ns;   --Etat actif du signal RESET
RESET <= '0';wait for 100 ns;

                wait; -- will wait forever
        END PROCESS;
END;
```

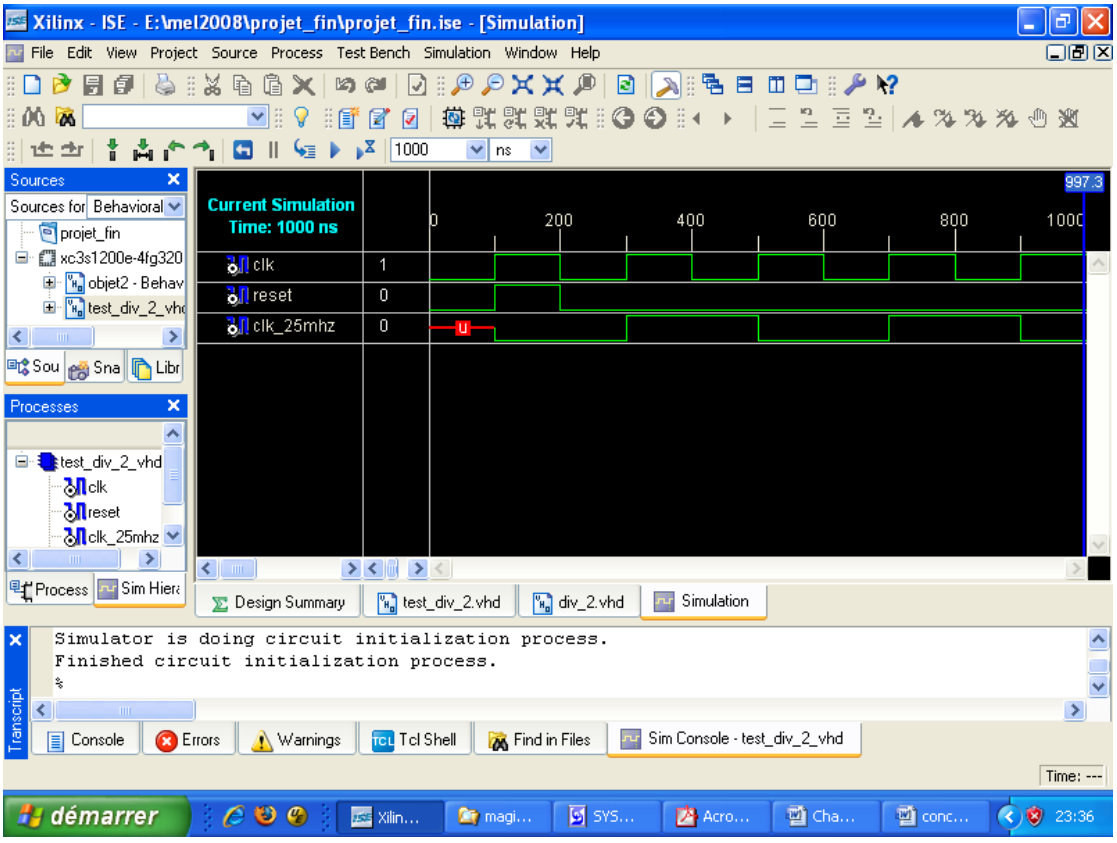


Figure 3.41 : Simulation avec *test Bensh* du bloc DIV_2.

Comme vous pouvez l'observer dans la figure 3.41, sur une échelle de 1000ns nous avons une horloge CLK d'une période de 200ns, à l'état actif du signal RESET aucun résultat. Nous

remarquons que la sortie CLK_25MHz de la bascule ne change d'état qu'au front montant du signal d'horloge CLK. Pour obtenir un cycle complet en sortie, il faut donc deux cycles complets du signal d'horloge. La fréquence du signal de sortie se trouve donc réduite de moitié.

Pour chaque synthèse, l'outil de synthèse doit analyser le code écrit de chaque bloc pour vérifier si la syntaxe est correcte. Si le code n'a aucune erreur syntactique, tout devrait fonctionner très bien. Autrement, une phase de correction des erreurs doit être effectuée.

Pour une réalisation physique, l'architecture doit être menée sur un dispositif tel qu'un FPGA, les signaux d'entrées-sorties doivent affectés aux différentes broches de ce dernier. Ceci peut être fait, avant la synthèse, en éditant un fichier de contraintes (*.ucf : *User Contraintes File*) ou bien graphiquement à l'aide de l'un des outils (PACE) du logiciel ISE (Annexe A). Voici le fichier des contraintes nécessaires pour la carte de développement NEXYS2 :

```
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
#PACE: Start of PACE I/O Pin Assignments
NET "CLK"          LOC = "B8" ;
NET "RESET"        LOC = "B18";
NET "D"            LOC = "D18" ;
NET "G"            LOC = "E18" ;
NET "SYNC_H"       LOC = "T4" ;
NET "SYNC_V"       LOC = "U3" ;
NET "JEU<0>"       LOC = "U5" ;
NET "JEU<1>"       LOC = "U4" ;
NET "JEU<2>"       LOC = "N8" ;
NET "JEU<3>"       LOC = "P8" ;
NET "JEU<4>"       LOC = "P6" ;
NET "JEU<5>"       LOC = "R9" ;
NET "JEU<6>"       LOC = "T8" ;
NET "JEU<7>"       LOC = "R8" ;
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
```

On remarque dans le fichier des contraintes présenté ci-dessus, le signal JEU sous forme d'un bus de 8 bits qui définit les trois couleurs RVB ; les deux premiers bits (JEU(0) et JEU(1)) présentent la couleur bleu, les trois bits suivants présentent la couleur verte, les trois derniers bits (JEU(5), JEU(6) et JEU(7)) présentent la couleur rouge.

On peut aussi observer le fichier des contraintes sur la figure 3.42 de l'environnement de XILINX ISE 9.2i, cette étape « *Assign Package Pins Post-Translate* » se trouve dans l'outil du placement et routage (*Implement Design*) :

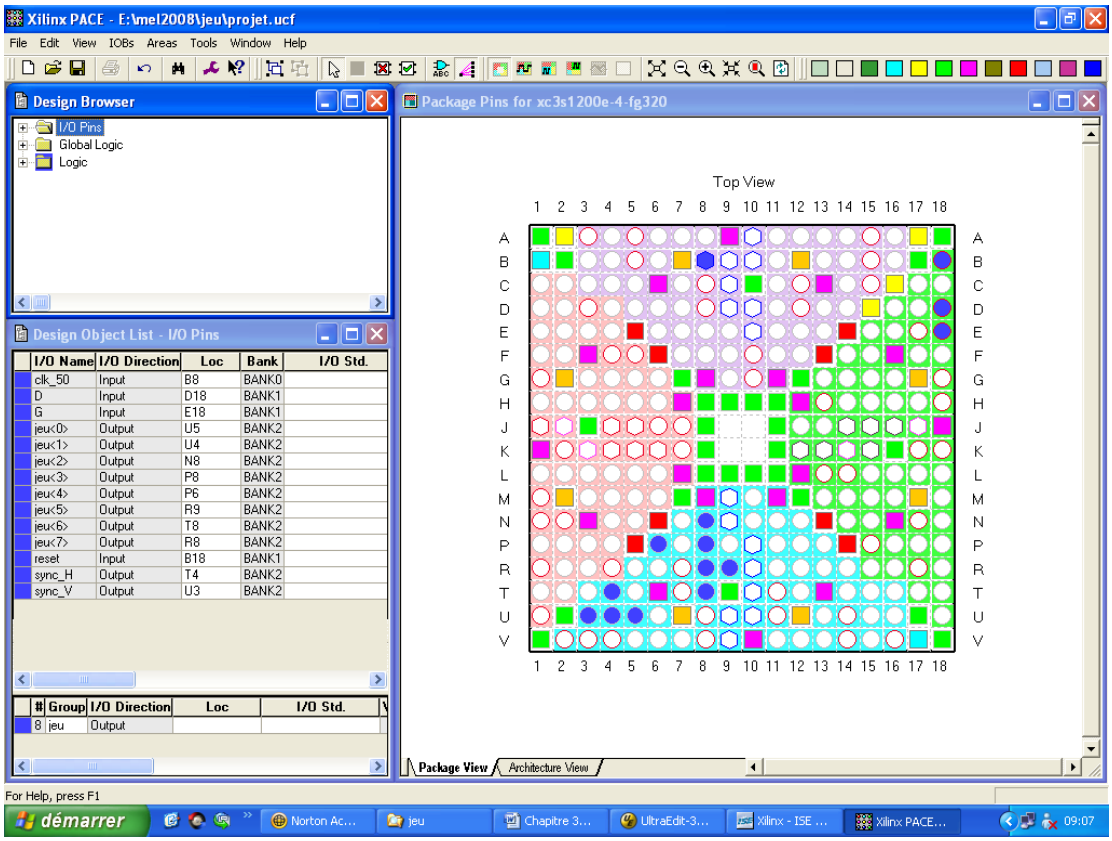


Figure 3.42 : Etape d'assignement des pins d'entrée/sortie.

La prochaine étape dans le processus est de générer le fichier de configuration (*.bit) en utilisant l'outil « *Generate Programming File* » du logiciel de XILINX ISE 9.2i (Annexe A). Ce fichier devra être téléchargé vers le composant reconfigurable « FPGA » en utilisant le câble USB2 par l'intermédiaire d'un outil propre de DIGILENT: ADEPT (Annexe C).

Enfin, nous avons fais la conception, la synthèse et l'implémentation des deux architectures sur la carte NEXYS2 à base d'un composant reconfigurable FPGA de 1200K portes. Le rapport de synthèse montre que certaines ressources sont particulièrement critiques.
Pour chaque conception, nous allons présenter le rapport de compilation (*View Design Summary*).

- **Pour la première conception :**

Tout d'abord la distribution logique (Tableau 3.3) ; nous constatons que nous utilisons pratiquement 1% des ressources disponibles :
1% des bascules.
1% des Lut à 4 entrées.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	74	17,344	1%
Number of 4 input LUTs	197	17,344	1%
Logic Distribution			
Number of occupied Slices	154	8,672	1%
Number of Slices containing only related logic	154	154	100%
Number of Slices containing unrelated logic	0	154	0%
Total Number of 4 input LUTs	252	17,344	1%
Number used as logic	197		
Number used as a route-thru	55		
Number of bonded IOBs	14	250	5%
IOB Flip Flops	1		
Number of GCLKs	3	24	12%
Total equivalent gate count for design	2,511		

Tableau 3.3 : Le sommaire d'utilisation de composant de la 1ère conception.

Nous pouvons noter que cette première architecture utilise un équivalent de 2511 portes logiques.
Le résumé (Tableau 3.4) indique toutes les données relatives au projet :

JEU_1 Project Status			
Project File:	jeu_1.ise	**Current State:**	Programming File Generated
Module Name:	objet2	**• Errors:**	No Errors
Target Device:	xc3s1200e-4fg320	**• Warnings:**	25 Warnings
Product Version:	ISE 9.2i	**• Updated:**	mar. 11. mai 06:48:05 2010

Tableau 3.4 : Le statut de la première conception.

- **Pour la deuxième conception :**

En changeant la conception, nous remarquons que le pourcentage d'utilisation des ressources disponible a augmenté (Tableau 3.5) :
1% des bascules.
2% des Lut à 4 entrées.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	92	17,344	1%
Number of 4 input LUTs	366	17,344	2%
Logic Distribution			
Number of occupied Slices	225	8,672	2%
Number of Slices containing only related logic	225	225	100%
Number of Slices containing unrelated logic	0	225	0%
Total Number of 4 input LUTs	400	17,344	2%
Number used as logic	366		
Number used as a route-thru	34		
Number of bonded IOBs	14	250	5%
IOB Flip Flops	1		
Number of GCLKs	3	24	12%
Total equivalent gate count for design	3,804		

Tableau 3.5 : Le sommaire d'utilisation de composant de la 2ème conception.

Nous pouvons constater que la deuxième architecture utilise un équivalent de 3804 portes logiques.
Finalement le résumé (Tableau 3.6) indique toutes les données relatives au projet :

JEU Project Status			
Project File:	jeu.ise	Current State:	Programming File Generated
Module Name:	jeu_video	• Errors:	No Errors
Target Device:	xc3s1200e-4fg320	• Warnings:	49 Warnings
Product Version:	ISE 9.2i	• Updated:	mer. 10. févr. 12:52:37 2010

Tableau 3.6 : Le statut de la deuxième conception.

Au lancement d'outil de vérification des erreurs, nous observons l'affichage de différentes alarmes (warnings), ces derniers permet par exemple d'identifier la présence des signaux déclarés mais non utilisés dans le programme. Cependant, la présence d'alarme n'empêche pas de poursuivre normalement les autres étapes du développement.

Nous remarquons que le nombre de portes utilisés dans les deux architectures précédentes (2511 et 3804 portes) est négligeable devant le nombre des portes qui existent dans le FPGA (1200 K portes). Mais en le comparant avec l'utilisation des circuits standards, par exemple il n'est pas négligeable devant 628 boitiers TTL de 4 portes chacun. Ce qui nous amène à choisir les composants de hautes densités d'intégration pour réaliser des circuits plus complexes avec une souplesse maximale, une rapidité en temps de conception.

Les deux architectures étaient implémentées dans notre circuit reconfigurable FPGA, et nous avons remarqué la différence entre eux dans l'affichage des formes des motifs utilisés dans l'application du jeu vidéo.

3.4 Conclusion :

A partir de la description VHDL, le passage à la synthèse représente une étape essentielle dans le processus de conception d'un composant programmable. Le résultat de cette synthèse va déterminer les caractéristiques du composant cible. Il est donc important et même crucial, dans la plupart des cas, de savoir orienter une synthèse de façon à optimiser le résultat. Mais optimiser ne signifie pas forcement diminuer le nombre de portes logiques équivalentes. On peut, par exemple, choisir de diminuer au maximum le nombre de bascules D au détriment du nombre de portes logiques ou encore privilégier certaines structures plutôt que d'autres, sachant que ces structures optimisent l'intégration dans le composant cible. Ceci étant, il est possible de créer deux descriptions équivalentes à la simulation dont la synthèse ne donnera pas le même résultat. Il y a donc tout un savoir-faire à acquérir pour rédiger une description VHDL de façon à orienter le résultat de la phase de synthèse.

La phase de conception été achevée, nous nous sommes attaché à la mise en place du système sur la carte de développement, et nous avons analysé les performances de notre solution. Si le projet peut sembler aisé, nous nous sommes vite rendu compte des difficultés inhérentes à la conception et la synchronisation des objets apparus sur le moniteur VGA.

Au terme de ce projet, nous estimons avoir satisfait les contraintes du cahier des charges dans son intégralité, à savoir concevoir une architecture matérielle en temps réel.

Conclusion et perspectives

Conclusion :

Les différents travaux effectués dans ce mémoire se sont déroulées sur l'évolution des circuits numérique, et en particulier sur les circuits programmables.

Pour tous les systèmes électroniques, la plus grande force de développement provient de l'accroissement constant des performances et des capacités logiques des nouvelles technologies de fabrication. En effet, lorsque les technologies de fabrication s'améliorent, les FPGA deviennent plus rapides, disposent d'une plus grande capacité et le coût par porte logique diminue. Ceci permet aux FPGA de prendre une part du marché qui était réservée auparavant aux implémentations sur ASIC.

Dès l'origine, les FPGA, avaient la réputation de mettre à disposition de l'utilisateur une conception rapide, fiable et simple en utilisant des langages de description de haut niveau qui ont pour avantage la possibilité de décrire des systèmes très complexes en quelques lignes de code. Cette méthodologie de conception reste absolument gérable et permet la réalisation d'applications très performantes moyennant une bonne connaissance des ressources offertes.

Tout au long de ce travail, nous avons confronté nos connaissances théoriques à la pratique, et par conséquent nous avons pu acquérir une double compétence : une bonne connaissance des progrès technologiques de la microélectronique dans les circuits numériques, et la maîtrise d'un outil de description abstrait : le langage VHDL. Ce dernier nous a permis de réaliser une architecture matérielle afin de l'implanter dans un circuit reconfigurable de type FPGA en respectant les différentes étapes de conception (conception à l'aide des outils de CAO), et avec une bonne dose de patience et de persévérance, nous avons abouti à un résultat final encourageant. Nous avons réalisé notre application qui consiste à concevoir un contrôleur vidéo implanté comme exemple sur un jeu vidéo. Cette application consiste à respecter des contraintes très précises, dont il faut générer les signaux de synchronisation avec un timing strict pour l'affichage d'une image.

Perspectives :

L'approche que nous avons développée présente certainement de nombreuses perspectives d'avenir intéressantes. Ces circuits reconfigurables FPGA ont permis déjà d'intégrer une foule d'opérations et de fonctions de plus en plus complexes. Le travail présenté dans ce mémoire est en phase exploratoire, permet de franchir plusieurs domaines en réalisant différentes applications plus complexes.

On peut aussi configurer une application dans un circuit programmable FPGA selon trois modes : la configuration statique, partielle ou dynamique, les deux derniers modes sont importants pour un futur travail afin de réaliser des circuits plus complexes et qui nécessitent un traitement en temps réel. Donc par exemple pour La reconfiguration dynamique (RTR: *Run Time Reconfiguration*) des FPGA est devenue un procédé de plus en plus utilisé par les systèmes reconfigurables appliqués à des domaines très divers. Dans le domaine du traitement

d'image, nous pouvons proposer un système de vision temps réel. Ce dernier permet l'implémentation d'algorithmes de traitements d'images bas niveaux. Nous pouvons enchaîner par exemple différents filtres sur la même image, permettant d'obtenir de meilleures performances.

La conception d'application à base de FPGA intéresse aussi de plus en plus les concepteurs des systèmes embarqués avec des systèmes sur puce (SOC : *System On Chip*), qui ont permis aux FPGA de ne plus être seulement utilisés pour des applications de types prototypages rapides ou la logique de glue (*Glue Logic*), mais comme des circuits essentiels dans la mise en œuvre des systèmes embarqués en intégrant des cœurs de processeurs, des blocs de propriétés intellectuelles IP.

ANNEXES

Annexe A. Tutorial du logiciel ISE de XILINX

ISE Foundation est développé et commercialisé par la société XILINX. Il permet de réaliser toutes les phases de conception de systèmes digitaux sur des composants reconfigurables (FPGA) de la société XILINX. Il est constitué d'une dizaine d'outils comme : *Project navigator, FloorPlanner, FPGA editor*.

Project navigator : outil de description du projet (spécification du système à concevoir en VDHL, schématique, machines d'états), synthèse, placement, routage.

FloorPlanner : outil de visualisation et de localisation des éléments utilisés du composant, ainsi que des communications réalisées.

FPGA editor : outil de visualisation du placement routage, disposant d'un placeur routeur intégré.

Dans cette annexe, nous allons introduire les notions fondamentales de ***Project navigator***, qui nous seront utiles pour la conception de notre application afin de l'implémenter sur le circuit reconfigurable FPGA

Ouvrir ISE :

Démarrer / Tous les Programmes / Xilinx ISE 9.2i / Project Navigator

Figure A.1 : Lancement du Project Navigator.

1°) Créez un nouveau projet :

File / New Project

Donnez un nom pour le projet et enregistrez-le dans un répertoire (Figure A.2).

Laissez ***Top-Level source type*** en **HDL.**
Cliquez sur ***Next.***

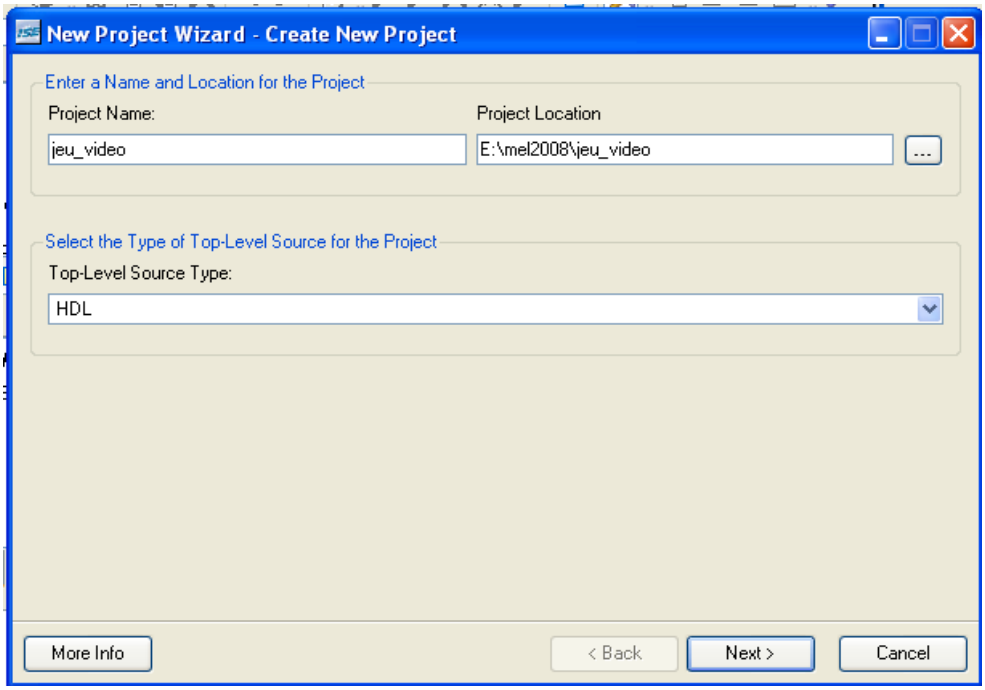


Figure A.2 : Création d'un projet.

La source est le fichier VHDL que va compiler le programme ISE pour créer le circuit électronique sur le FPGA de la carte Spartan3E (NEXYS2). Nous utilisons le langage VHDL mais d'autres langages peuvent être utilisés comme VERILOG. Nous travaillerons à partir de fichiers écrits en VHDL donc le type est HDL. D'autres types sont possibles par exemple le type schématique (*schematic*) pour lequel on définit son système à partir de schémas de composants qu'on assemble. Le logiciel décrypte alors directement le schéma.
Le terme *Top-level* indique que le fichier utilisé est celui qui sera compilé et implanté sur le FPGA. D'autres fichiers seront utilisés comme sous-programmes.

2°) Remplissez les catégories suivantes :

Pour une carte Spartan 3E

- ✓ Product Category : **All**
- ✓ Family : **Spartan3E**
- ✓ Device : **XC3S1200E**
- ✓ Package : **FG320**
- ✓ Speed Grade : **-4**
- ✓ Top-Level Module Type : **HDL**
- ✓ Synthesis Tool : **XST (VHDL/Verilog)**
- ✓ Simulator : **ISE Simulator (VHDL/Verilog)**
- ✓ Preferred language : **VHDL**

Vérifiez que : ***Enable Enhanced Design Summary*** est coché.

Laissez les valeurs par défaut dans les champs restant (Figure A.3).
Cliquez sur ***Next.***

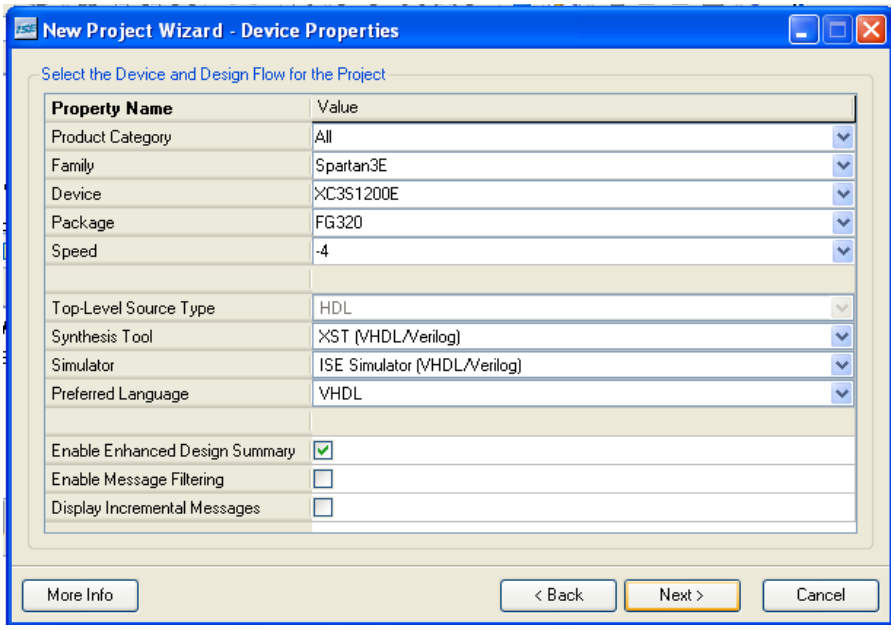


Figure A.3 : Propriétés de la carte Spartan3E utilisée.

3°) Créez une source VHDL :

- ✓ Dans la fenêtre ***New Project Wizard*** cliquez sur ***New Source.***
- ✓ Sélectionnez le type de source : **VHDL module** (Figure A.4).
- ✓ Entrez un nom pour le fichier.
- ✓ Vérifiez que : ***Add to project*** est bien coché.
- ✓ Cliquez sur ***Next.***

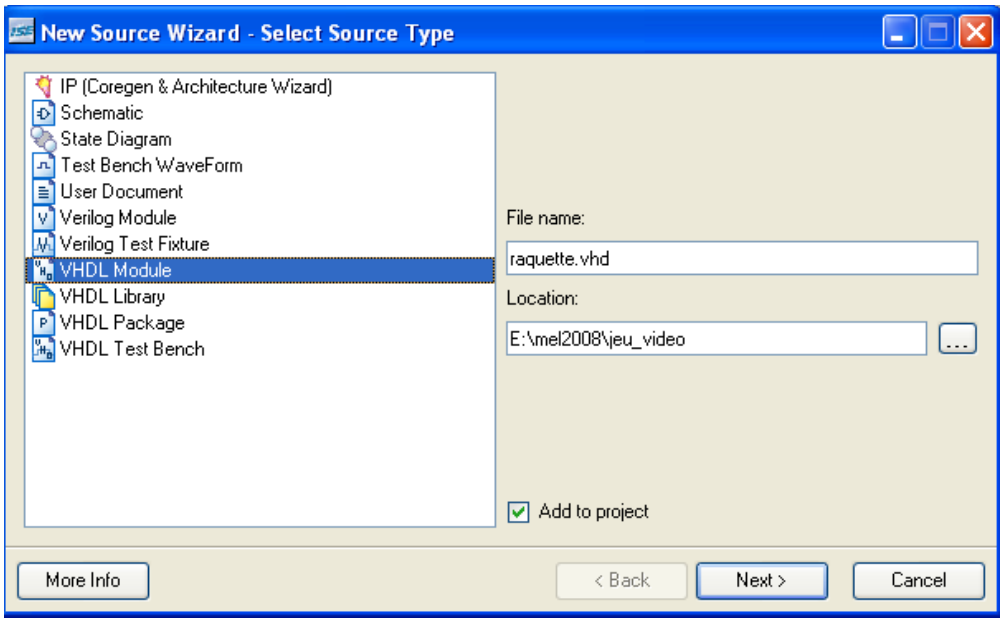


Figure A.4 : Création d'un fichier source VHDL.

4°) Définissez les entrées et sorties du programme :

Après le nom de l'***Entity***, vous pouvez choisir le nom de l'architecture (par défaut c'est ***behavioural***). Ce nom n'est pas important mais peut vous permettre de mieux vous repérer. Vous devez introduire les entrées et les sorties du module VHDL (Figure A.5).

- ✓ clk_1hz : in
- ✓ reset : in
- ✓ sync_v : in
- ✓ D : in
- ✓ G : in
- ✓ a : out (sur 10 bits)
- ✓ b : out (sur 10 bits)

Ici nous utilisons des entrées sur 1 bit et des vecteurs de 10 bits pour les sorties, donc vous devrez cocher **Bus** puis indiquer 9 dans **MSB** et 0 dans **LSB**.

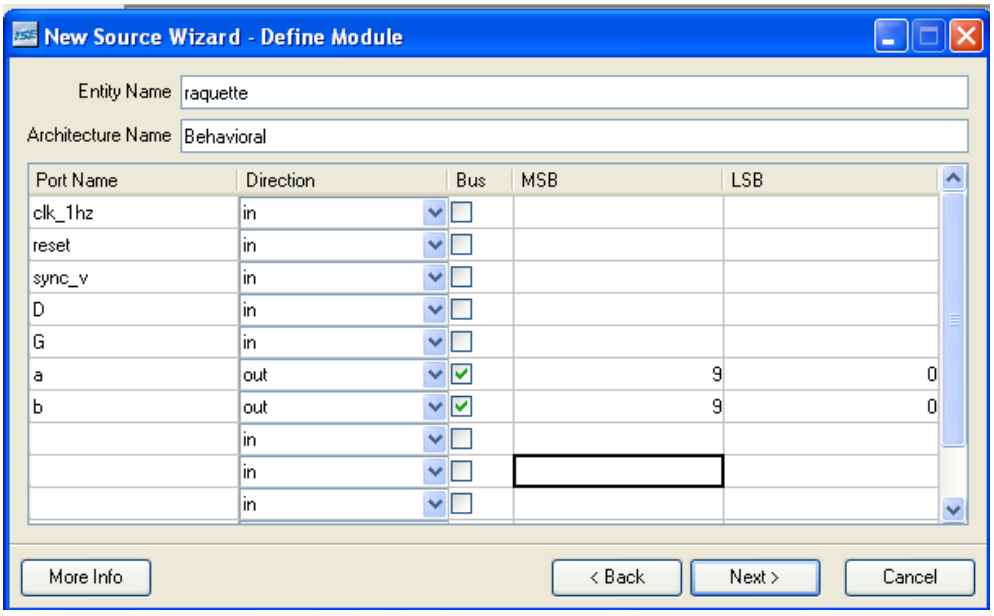


Figure A.5 : Définition des entrées et sorties.

Cliquez sur ***Next.***
Un résumé s'affiche, cliquez sur ***Finish*** (Figure A.6).

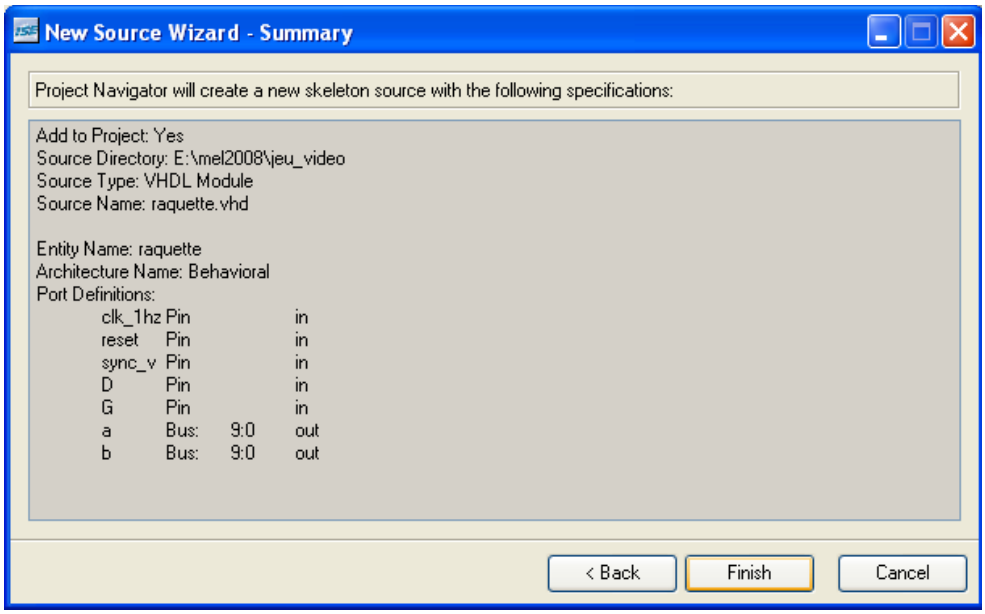


Figure A.6 : La création d'une source VHDL.

La partie création d'une source est terminée. Le logiciel va créer le corps du fichier VHDL que vous devez compléter :

```
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-- Module Name:    raquette - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity raquette is
    Port ( clk_1hz : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           sync_v : in  STD_LOGIC;
           D : in  STD_LOGIC;
           G : in  STD_LOGIC;
           a : out  STD_LOGIC_VECTOR (9 downto 0);
           b : out  STD_LOGIC_VECTOR (9 downto 0));
end raquette;

architecture Behavioral of raquette is

begin
```

```
-------------------------------------------------
à compléter
-------------------------------------------------
end Behavioral;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Après avoir compléter le fichier VHDL, il faut le sauvegarder puis examinez la syntaxe pour déceler les erreurs en cliquant dessus ***Check Syntax*** qui se trouve dans la fenêtre de ***processes.***

5°) Add existing source :

Vous avez la possibilité d'ajoutez des sources VHDL déjà existantes.

- ✓ Cliquez sur ***Add Source.***
- ✓ Sélectionnez le fichier VHDL.
- ✓ Vérifiez que ***Copy to project*** est bien coché.
- ✓ Cliquez sur ***Next.***
- ✓ Un résumé du projet s'affiche. Cliquez sur ***Finish***.

Une fenêtre vous demande de vérifier le statut du fichier VHDL que vous ajoutez au projet.

- ✓ Sélectionnez ***Synthesis/Imp + Simulation***.
- ✓ Cliquez sur ***OK.***

6 °) Définir le programme principal

Le programme principal est un fichier VHDL, qui fait appel aux sources déjà compilées (instancier les modules VHDL dans le fichier principal), il faut qu'il apparaisse comme ***Top-level*** (le fichier principal sera identifier dans la fenêtre ***Source*** par le motif de 3carrés). On obtient ceci en cliquant sur le fichier principal du bouton droit ***Set as Top module***. Par la suite, il faut synthétiser le fichier sur ***Synthesize***.

7°) Assignement des pins

Il vous faut maintenant définir les pins entrées/sorties du FPGA à utiliser. Comme le FPGA est intégré sur une carte comportant des afficheurs, des boutons poussoirs, une horloge, un port VGA, etc. la documentation de la carte fournit le nom des pins du FPGA reliées à ces entrées et sorties.

- ✓ Sélectionnez le fichier principal dans la hiérarchie
- ✓ puis dans la fenêtre ***Processes*** double cliquez sur ***Users Constraints / Assign Package Pins***.
- ✓ Une fenêtre ***Xilinx Pace*** s'ouvre (Figure A.7).

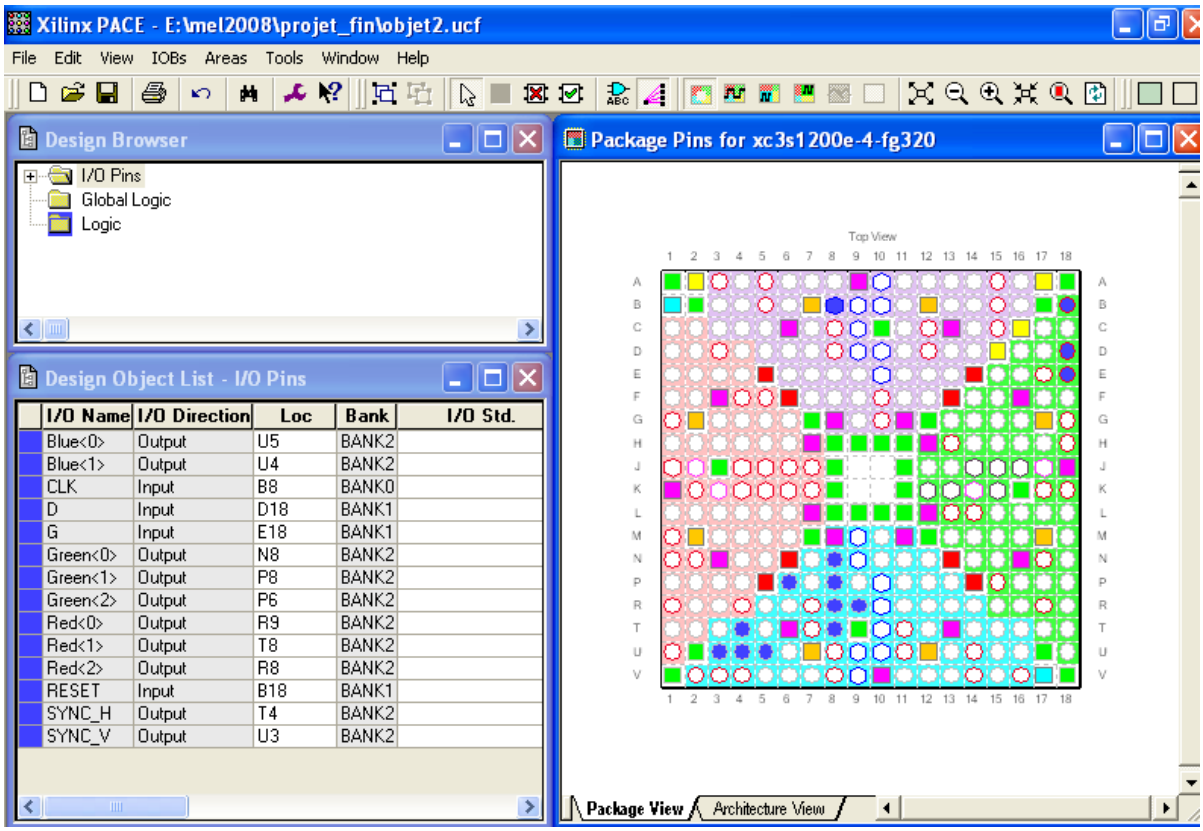


Figure A.7 : Assignement des pins d'entrées/sorties.

- ✓ A gauche dans ***Design Object List I/O pins*** entrez le nom des pins pour toutes les entrées et sorties dans la fenêtre ***Loc***.
- ✓ Enregistrez. Dans ***Bus delimiter*** sélectionnez ***XST Default***.
- ✓ Fermez la fenêtre ***Pace***.

8°) Pour programmer le FPGA de la carte :

Effectuez la synthèse en cliquant sur ***Synthesize-XST*** puis le placement-routage en double-clique sur ***Implement Design***. Et pour terminer il faut cliquer ***Generate Programming File*** afin d'avoir dans le projet un fichier (*.bit) pour programmer le FPGA**.**

Annexe B. les descriptions schématiques des éléments de la carte NEXYS2 de XILINX.

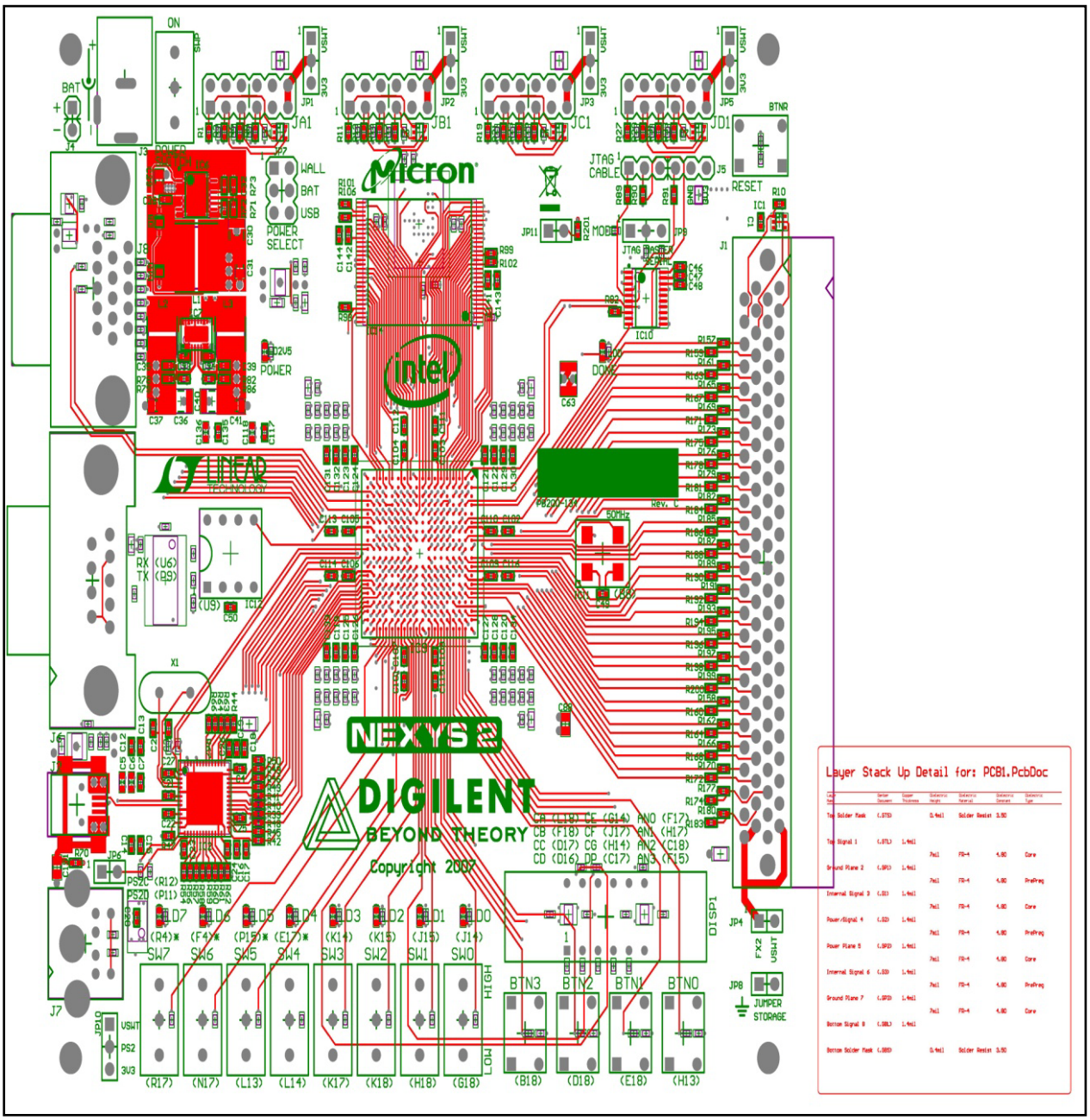


Figure B.1 : L'architecture globale de la carte NEXYS2.

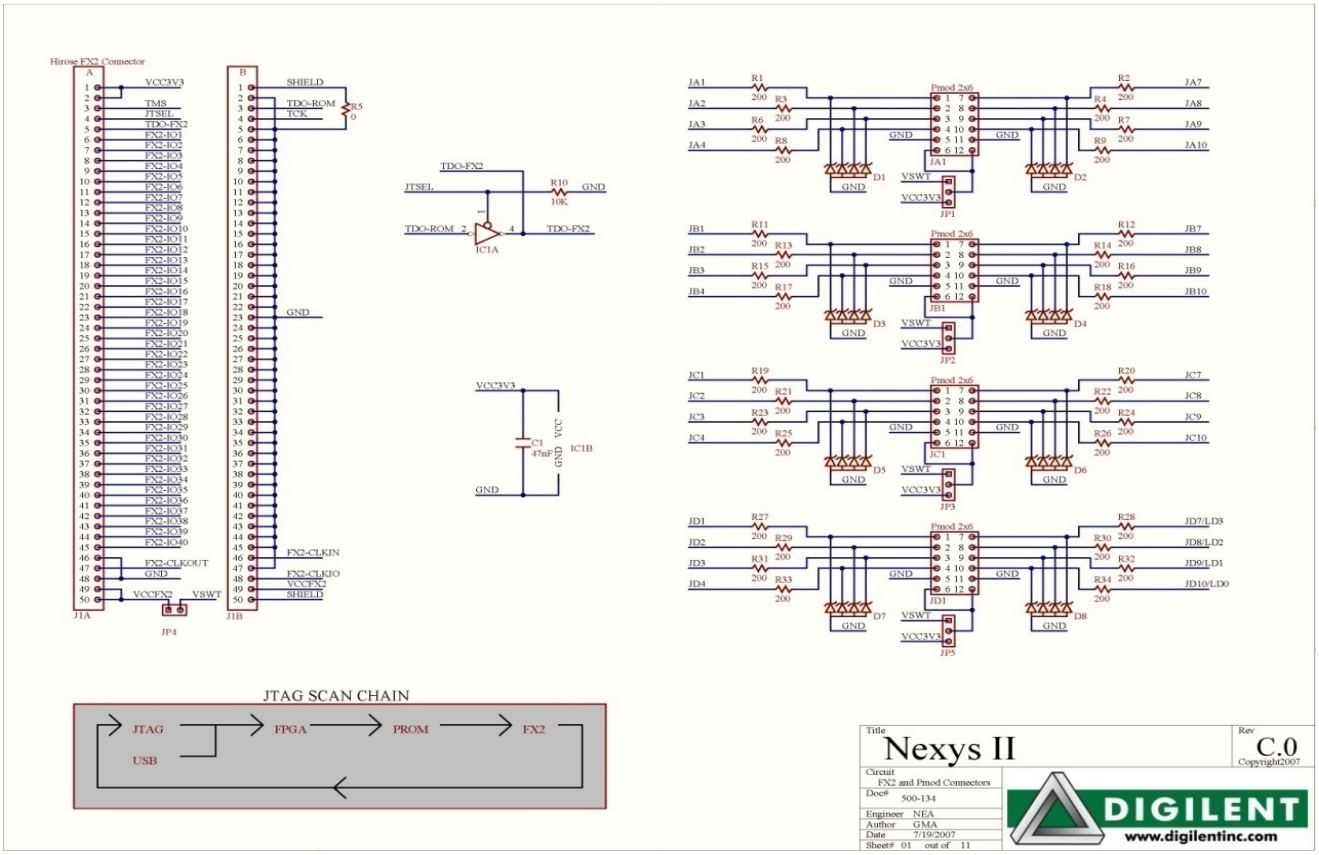


Figure B.2 : Schéma des connecteurs FX2 et Pmod.

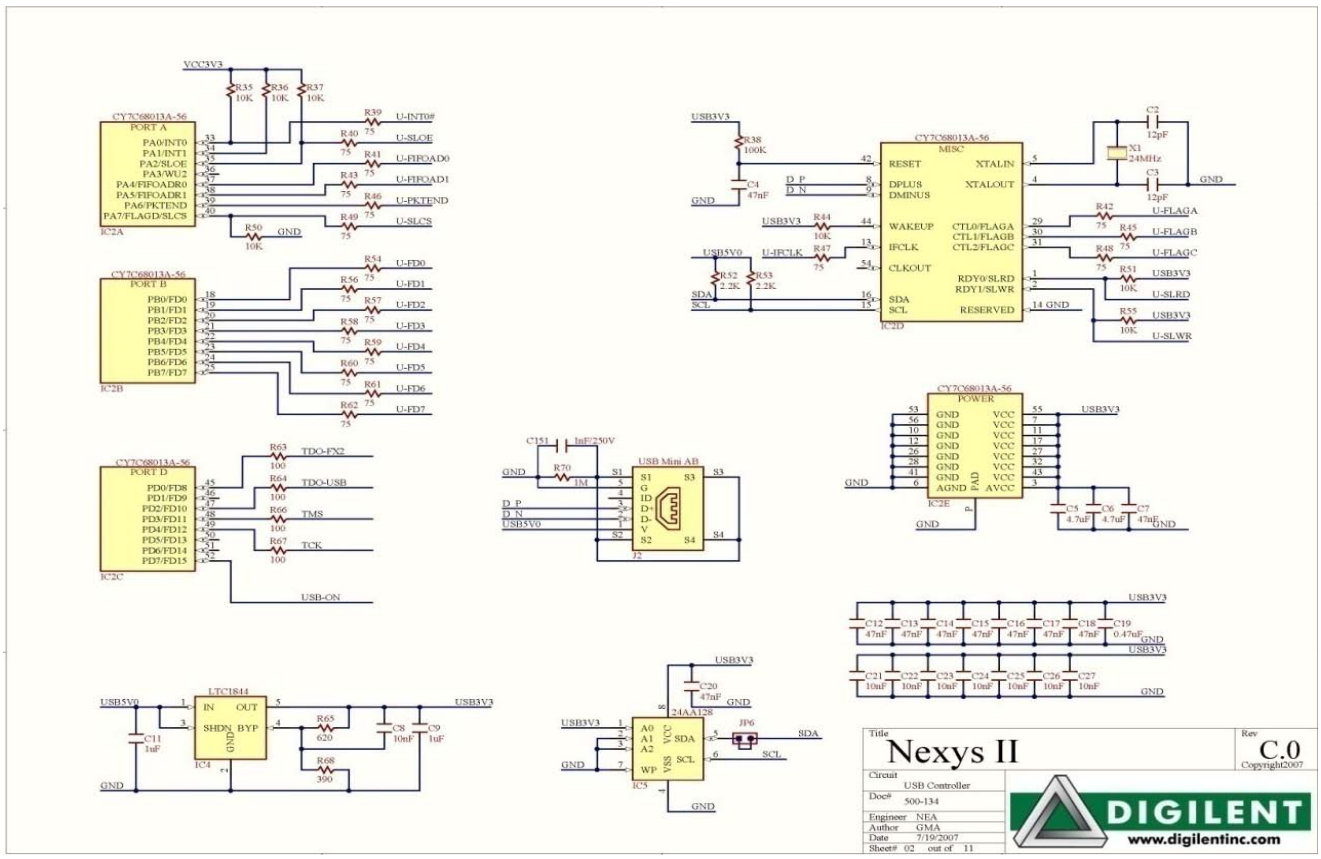


Figure B.3 : Schéma du contrôleur USB.

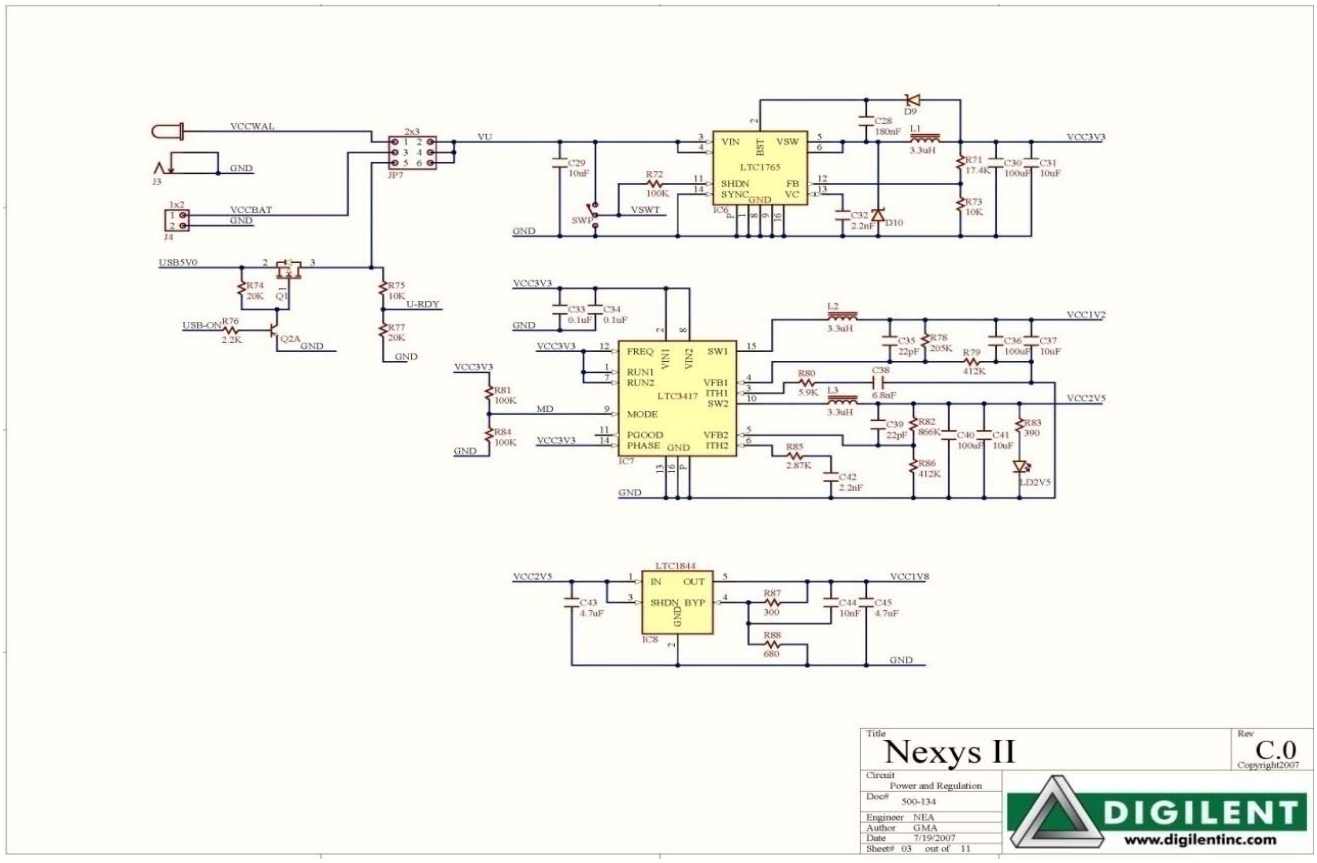


Figure B.4 : Schéma des circuits de la puissance et de la régulation.

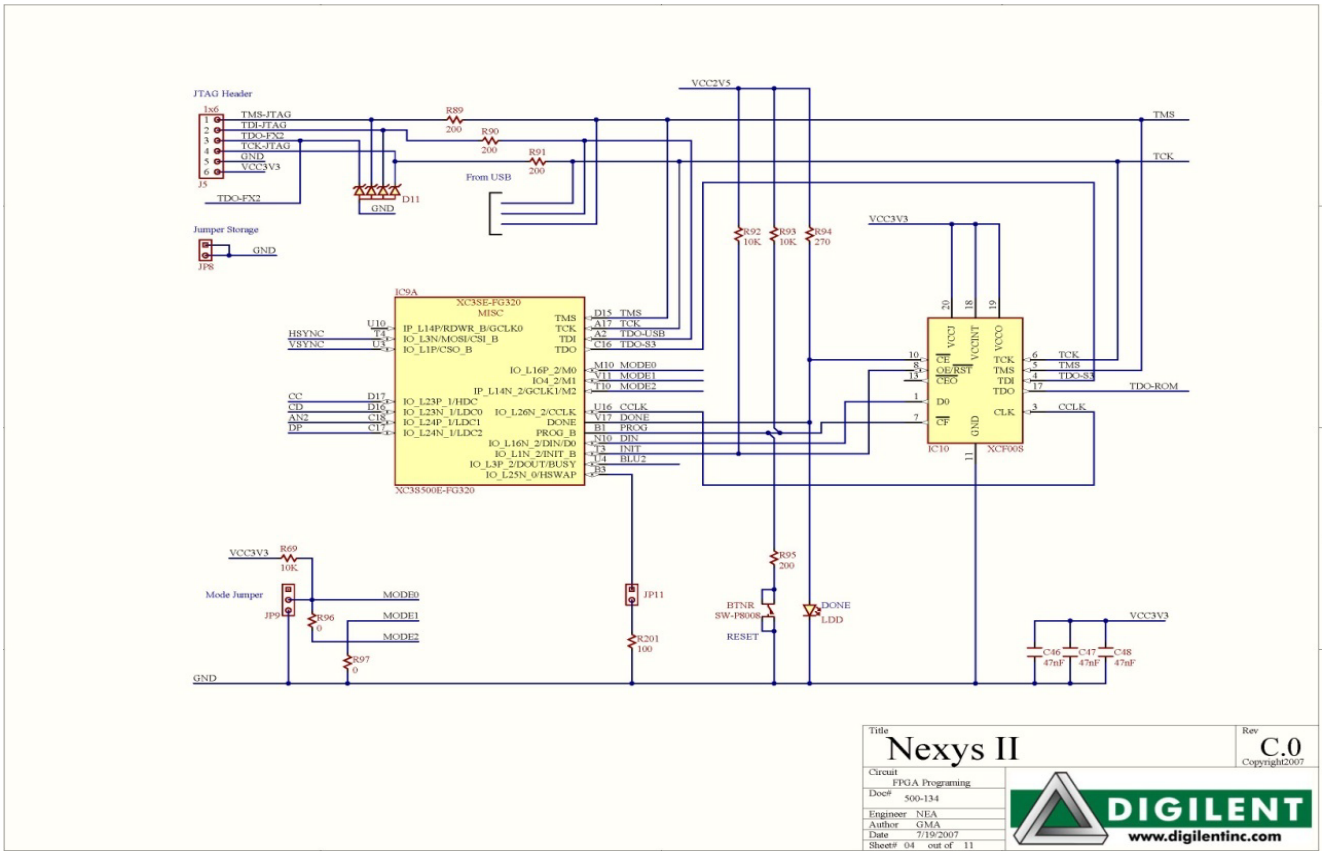


Figure B.5 : Schéma du circuit de la programmation du FPGA.

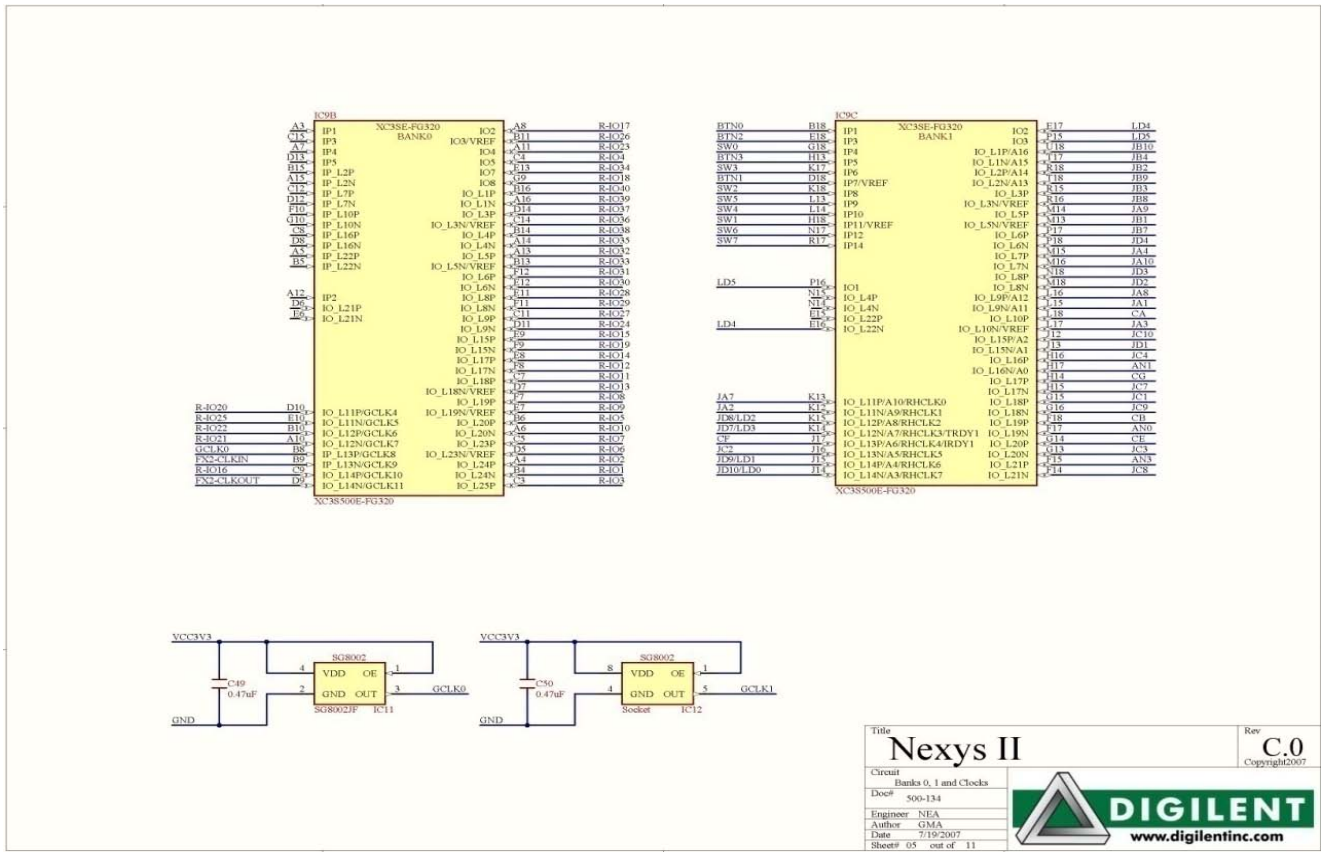


Figure B.6 : Schéma des Banks 0, 1 et d'horloge.

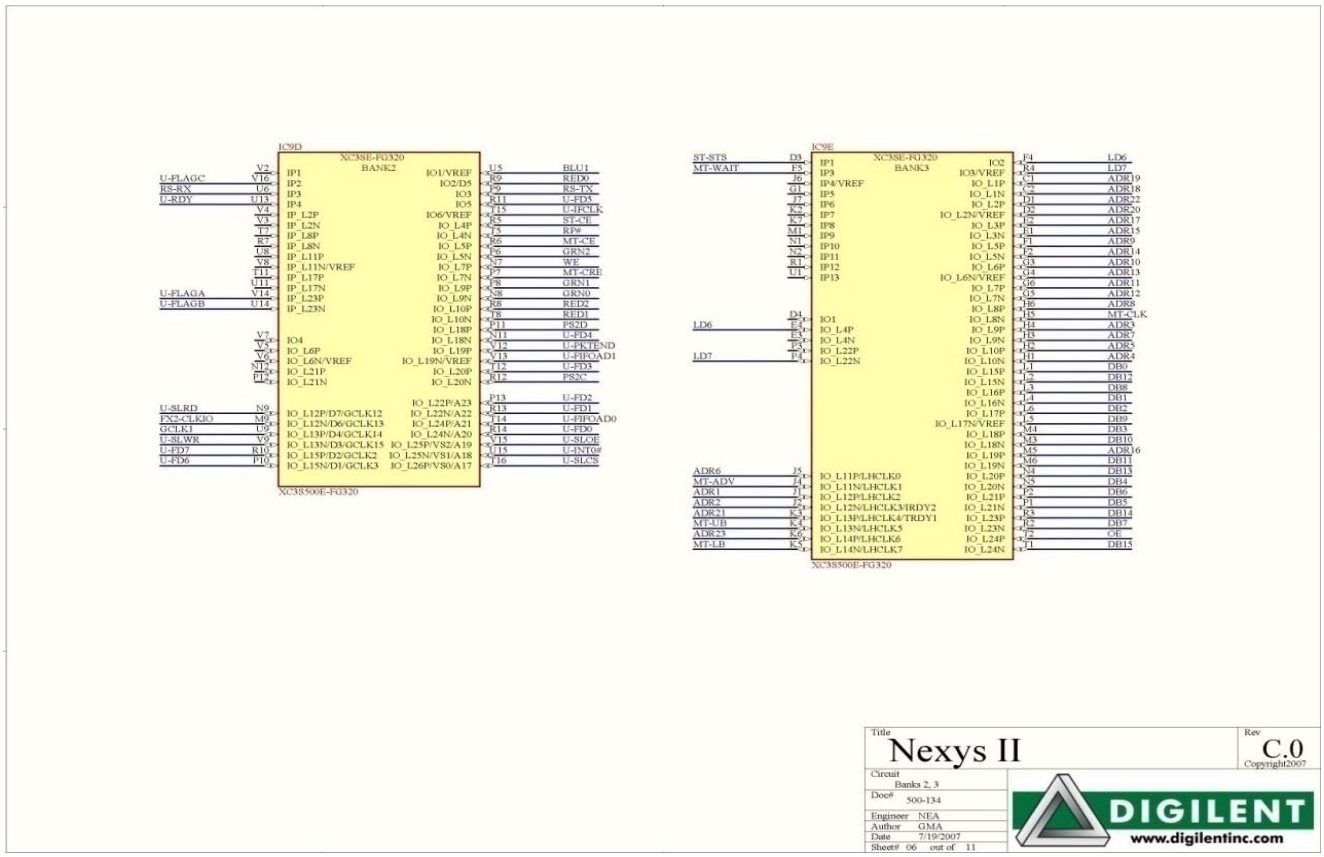


Figure B.7 : Schéma des Banks 2, 3.

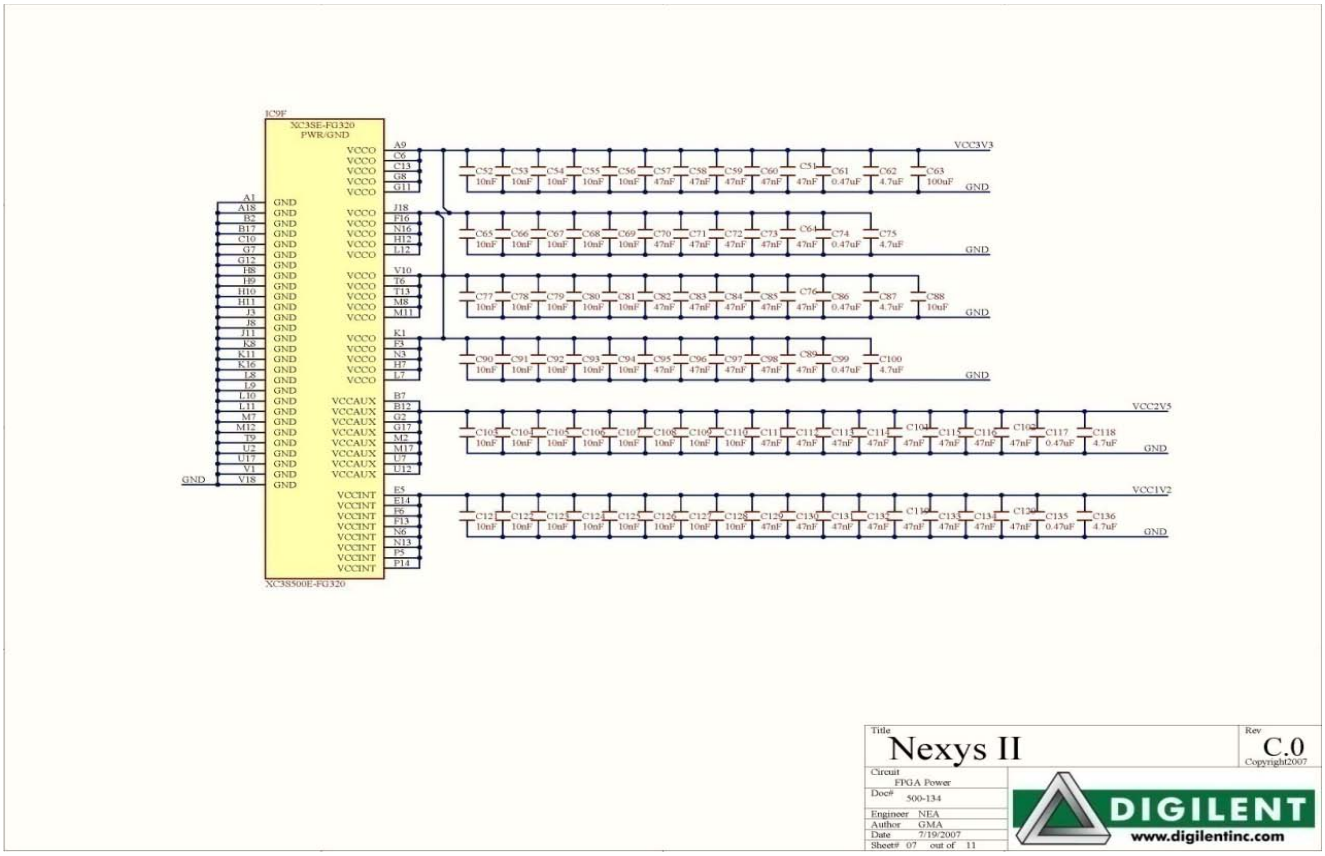


Figure B.8 : Schéma du circuit de la puissance du FPGA.

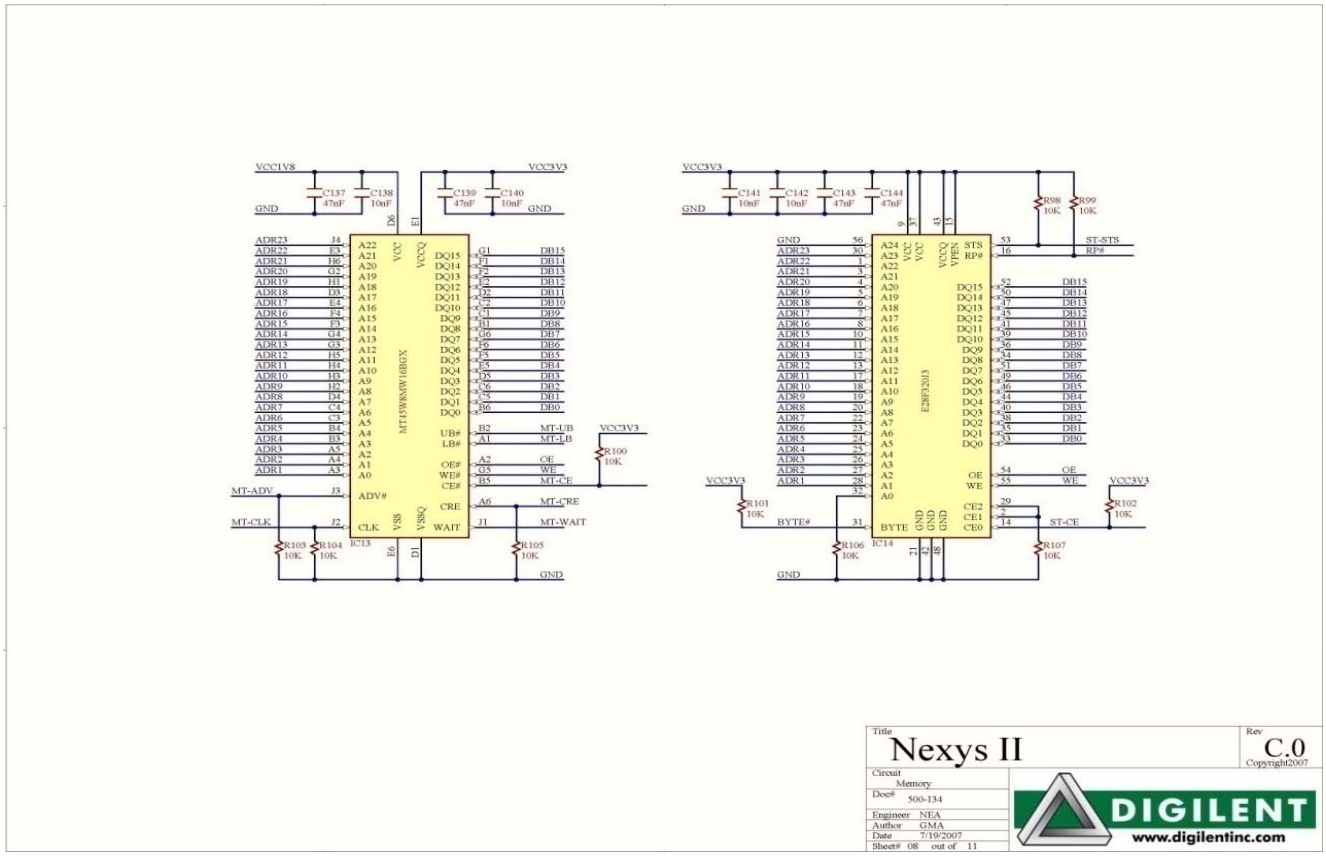


Figure B.9 : Schéma des mémoires.

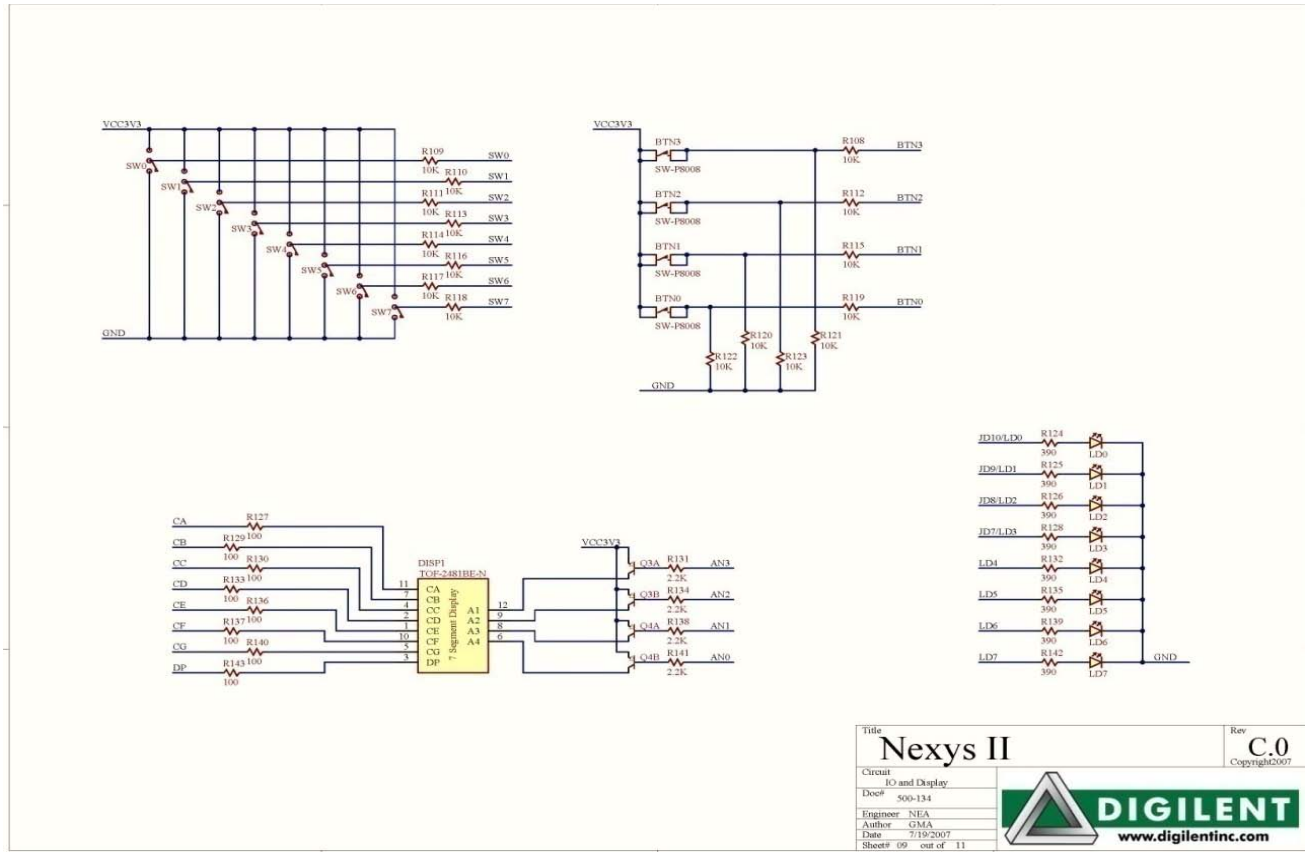


Figure B.10 : Schéma des circuits d'entrées/sorties et d'affichage.

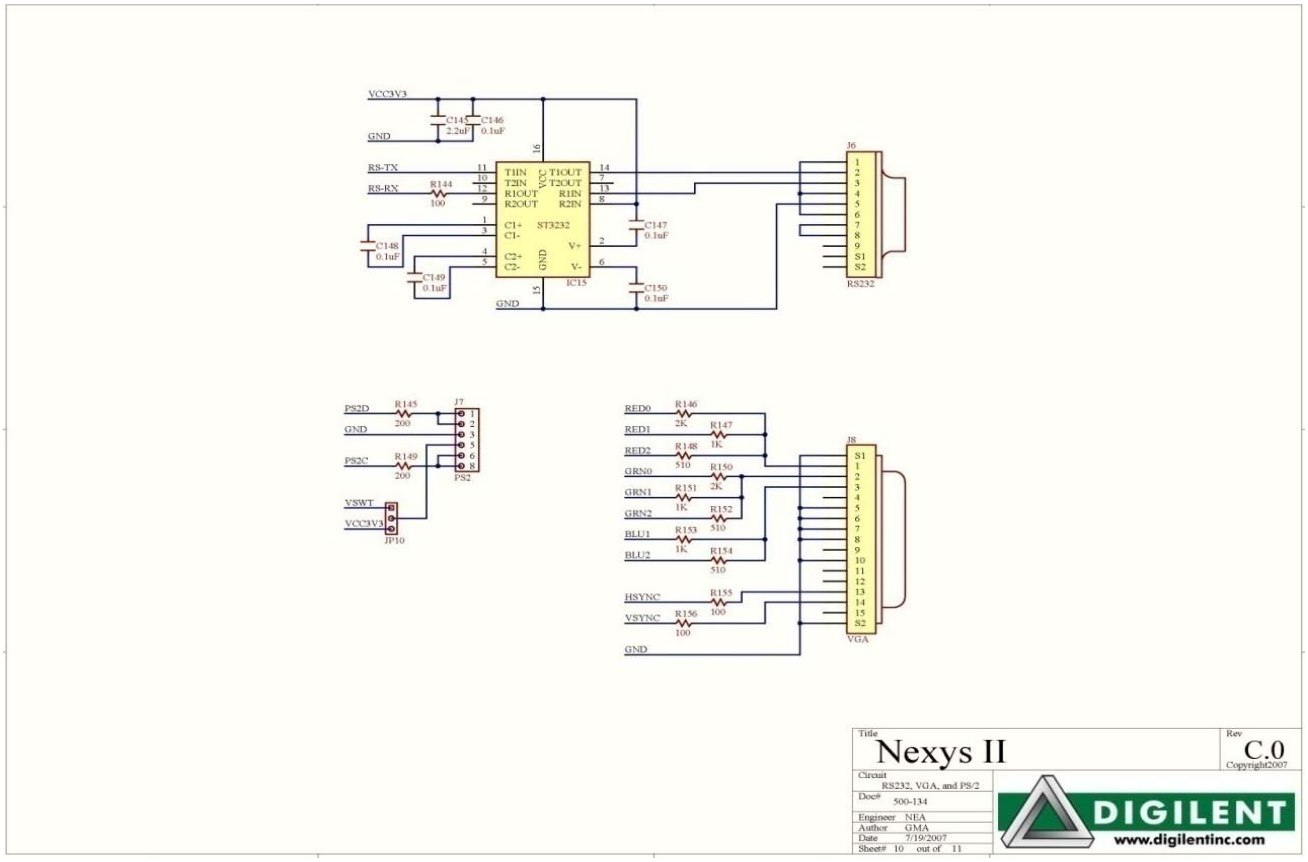


Figure B.11 : Schéma des circuits RS232, VGA et PS/2.

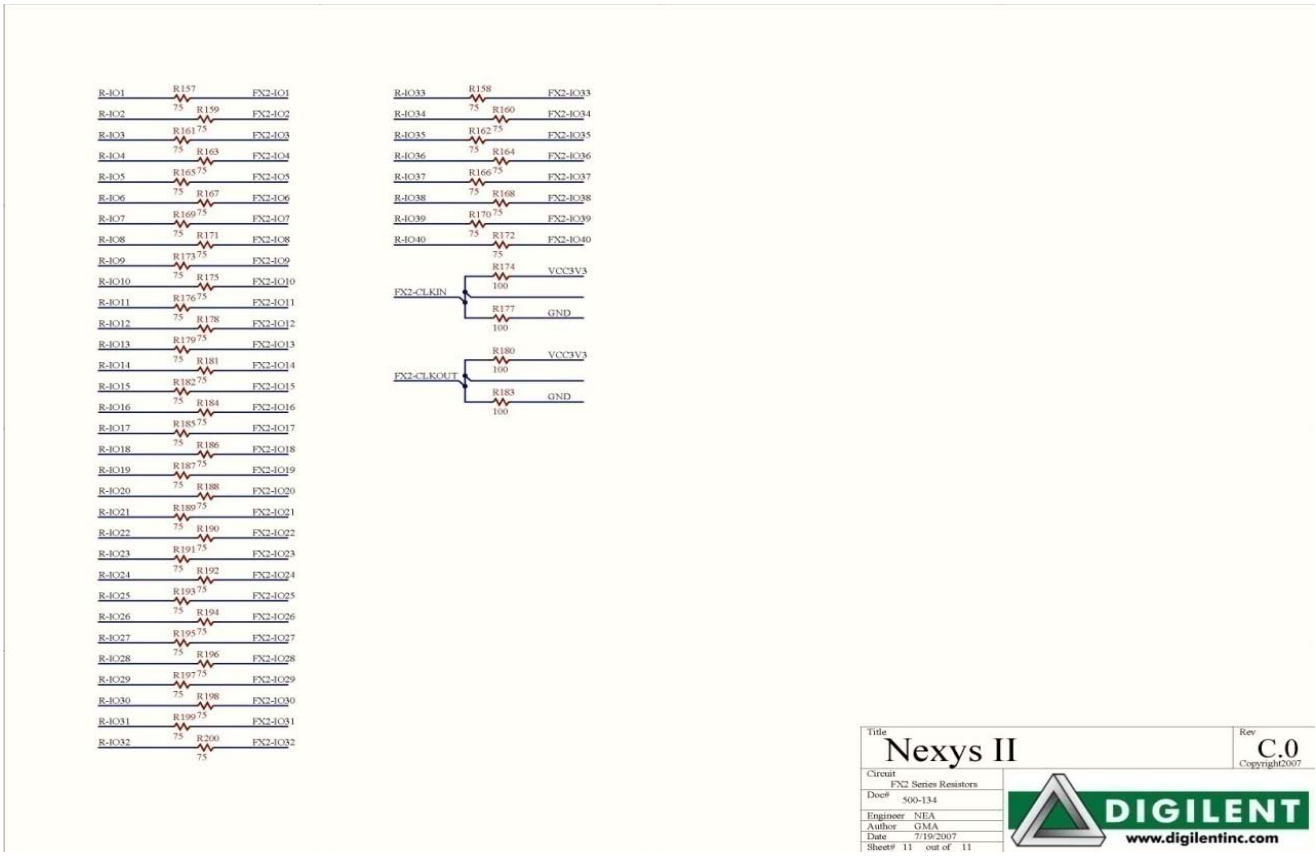


Figure B.12 : Schéma des résistances en séries de FX2.

Annexe C. Tutorial du l'outil ADEPT (propriétaire de DIGILENT)

Maintenant que le fichier de programmation a été produit (*.bit), il est temps de charger la conception dans le FPGA, et voir les résultats.
Les informations sont envoyées au FPGA via un câble reliant l'ordinateur à une plate forme comportant le FPGA et d'autres éléments tels que boutons poussoirs, connecteur VGA… Ceci peut se faire de différentes manières selon vos équipements disponibles et préférences. Dans cette annexe, nous discuterons en particulier de la configuration par le port USB qui était utilisé dans notre travail.

- ✓ Configuration en utilisant le port JTAG.
- ✓ Configuration en utilisant le port USB.

Configuration en utilisant le port USB et l'outil ADEPT :

La configuration du FPGA en utilisant le câble USB2 (Figure C.1) est faite par l'intermédiaire d'un outil propre de DIGILENT (**ADEPT)**. Ceci peut être téléchargé du web site de DIGILENT.

Figure C.1 : Câble de connexion USB2.

Ouvrir ADEPT :

Démarrer / Programmes / Digilent / Adept / Export

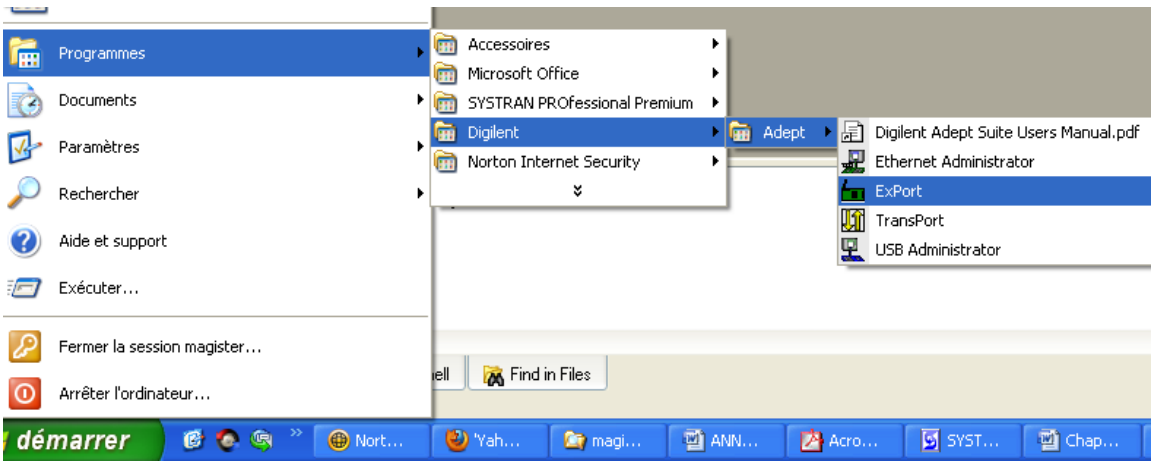


Figure C.2 : Lancement d'outil ADEPT.

- ✓ Vérifier que la connexion du PC et la carte numérique NEXYS2 par le câble USB2 est établit.
- ✓ mettre la carte numérique sous tension et assurez-vous que le ***jumper*** de mode (JP9) est placé dans la position de la " **ROM** ".
- ✓ Cliquez sur le bouton ***Initialize Chain*** afin d'identifier les composants de la plate forme (Figure C.3).

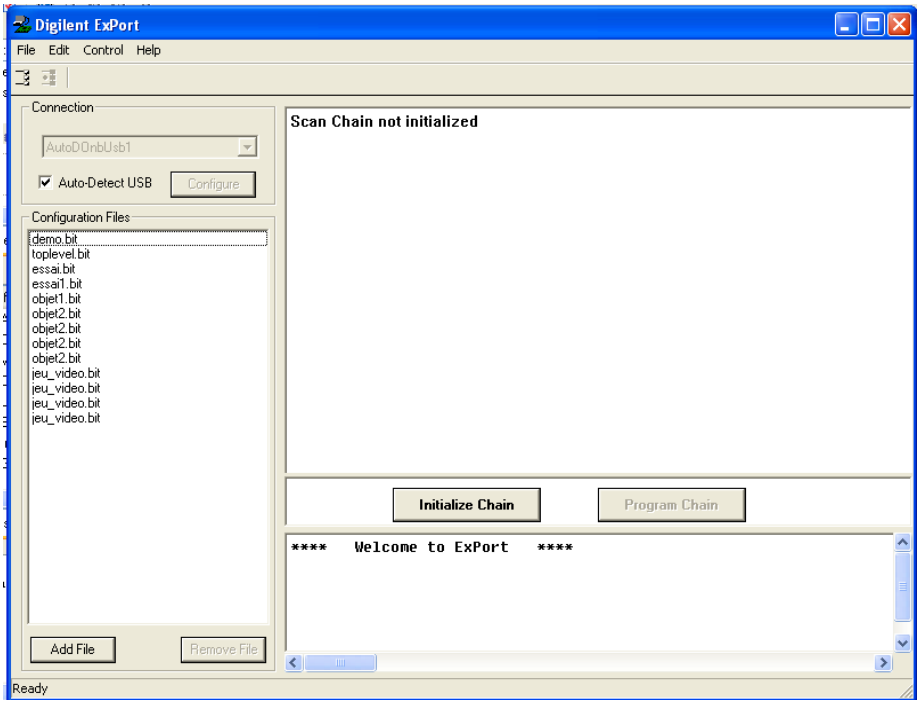


Figure C.3 : Initialisation de l'outil.

Une autre fenêtre apparait (Figure C.4), on trouve les composants FPGA, ROM de notre plate forme.

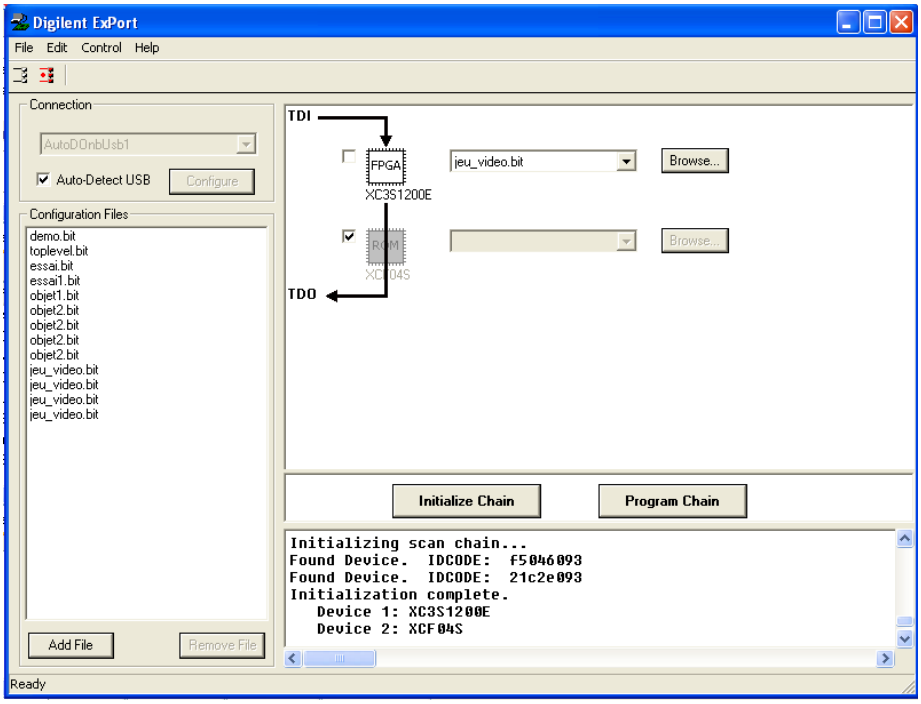


Figure C.4 : La recherche du fichier de configuration.

- ✓ Cocher la ROM.
- ✓ Chercher le fichier de configuration (*.bit) dans le projet en cliquant sur le bouton ***Browse***.
- ✓ Appuyer sur le bouton ***Program Chain*** (Figure C.5) afin de programmer le circuit reconfigurable FPGA.

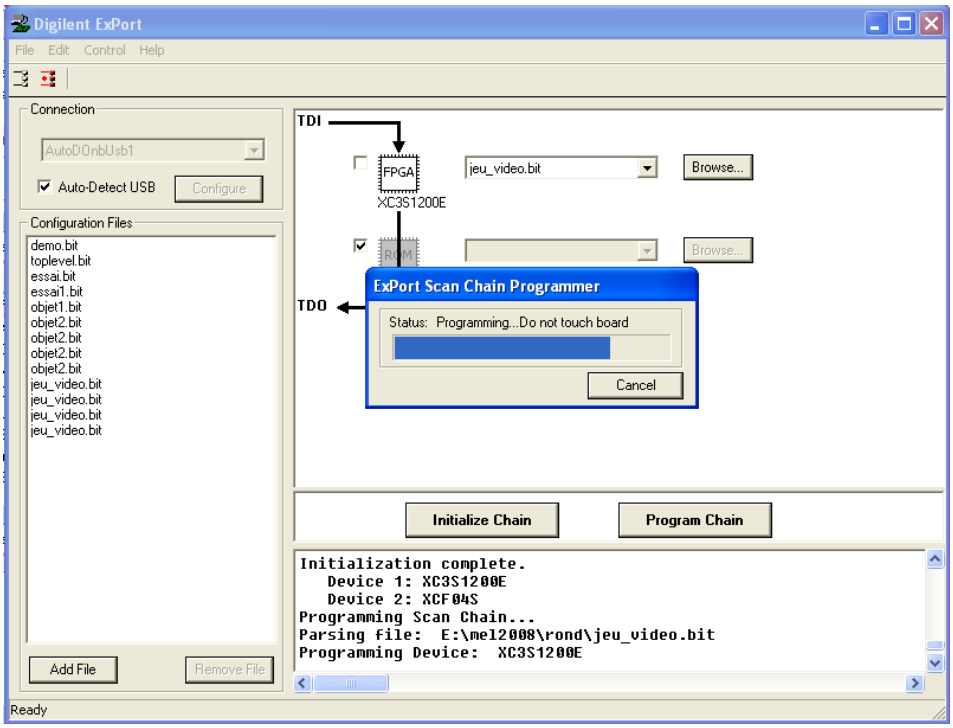


Figure C.5 : La configuration du FPGA.

Un message apparait indique que la configuration de votre FPGA est complète (Figure C.6). On constate le résultat de l'application à travers les sorties proposées (par exemple, on peut avoir le résultat sur un moniteur VGA à travers les sorties du connecteur VGA posé dans la plate forme).

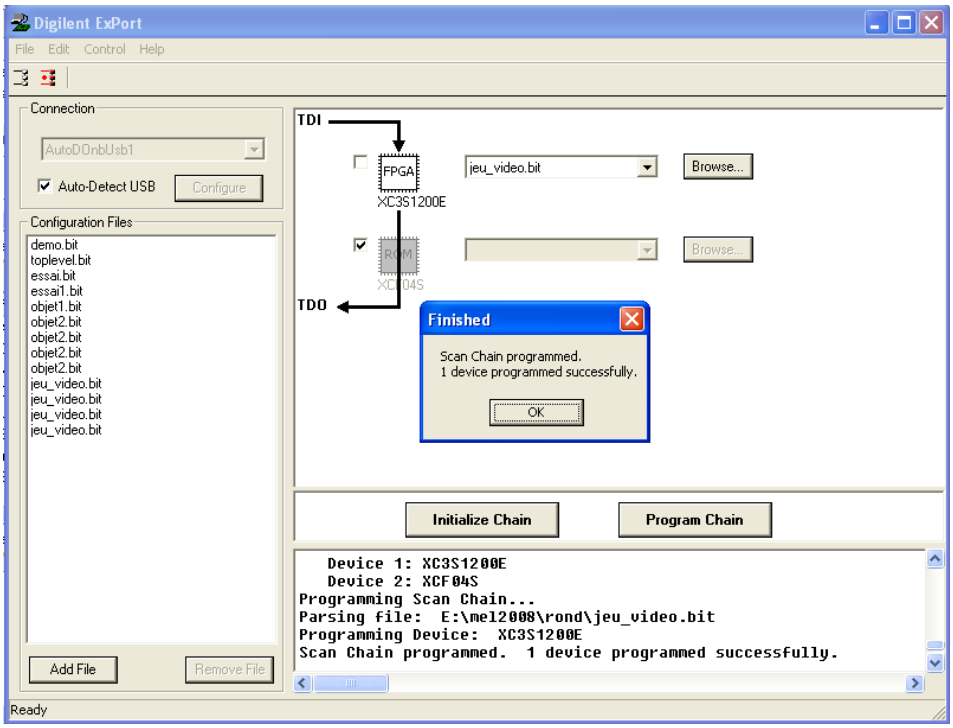


Figure C.6 : L'étape finale de configuration.

Glossaires

ABEL : *Advanced Boolean Equation Logical.*

ASIC : *Application Specific Integration Circuit.*

CAO : Conception Assistée par Ordinateur.

CLB : *Configurable Logic Blocks.*

CMOS : *Complementary Metal-Oxide Silicon.*

CNA : Convertisseur Numérique-Analogique.

CPLD : *Complex Logic Programmable Device.*

CSP : *Cross Search pattern.*

DAC : *Digital to Analogue Converter.*

DSP : *Digital Signal Processor.*

EDA : *Electronic Design Automation.*

EPROM: *Erasable Programmable Read-Only Memory.*

FPGA : *Field Programmable Gate Arrays.*

GAL : *Generic Arrays Logic.*

HDL : *Hardware Description Langage.*

IOB : *Input/Output Blocks.*

IP : *Intellectual Property.*

JTAG : *Joint Test Action Group.*

LAB : *Logic Array Block.*

LCA : *Logic Cell Array.*

LSB : *Least Significant Bit.*

LUT : *Look Up Table.*

MAC : *Multiplier and Accumulator.*

MOS : *Metal oxyd semiconductor.*

MSB : *Most Significant Bit.*

MSI : *Medium Scale Integration.*

OLMC : *Output Logic Macro Cell.*

PAL : *Programmable Array Logic.*

PI : *Programmable Interconnect.*

PLA : *Programmable Logic Arrays.*

PLD : *Programmable Logic Device.*

PROM : *Programmable Read Only Memory.*

RAM : *Random Access Memory.*

RGB : *Red, Green, Blue.*

ROM : *Read Only Memory.*

RTL : *Register Transfer Logic.*

RTR : *Run Time Reconfiguration*

RVB : Rouge, Vert et Bleu.

SOC : *System On Chip.*

SRAM : *Static Random Access Memory.*

SSI : *Small Scale Integration.*

TTL : *Transistor Transistor Logic.*

USB : *Universal Serial Bus.*

VGA : *Video Graphics Adapter.*

VHDL : *Very High Speed Integrated Circuit Hardware Description Language.*

VLSI : *Very Large Scale Integration.*

Bibliographie

LIVRES.

[NKETSA-98] Alexendre NKETSA, «**Circuits Logiques Programmables, Mémoires, PLD, CPLD et FPGA», ***Ellipses, Paris, 1998, ISBN 2-7298-6792-9.*

[DUTRIEUX-97] Laurent DUTRIEUX & Didier DEMIGNY**, «Logique Programmable, Architecture des FPGA et CPLD», ***Edition Eryrolles, 1997, ISBN 2-212-09581-3.*

[ANCEAU-07] Francois ANCEAU & Yvan BONNASSIEUX**, «Conception Des Circuits VLSI, Du composant au système », ***Dunod, 2007, ISBN 978-2-10-050036-9.*

[WEBER-07] Jacques WEBER & Sébastien MOUTAULT & Maurice MEAUDRE**, « Le langage VHDL, du langage au circuit, du circuit au langage »**, *Dunod, Paris, 2007, ISBN 978-2-10-050191-5.*

[DARCHE-04] Phillip DARCHE, **« Architecture Des Ordinateurs, Logique booléenne : implémentations et technologies », ***Vuibert, Paris, 2004, ISBN 2-7117-4821-9.*

[HACK-94] U.HACK & M.HOFFMANN**, « Le Manuel Des Gal », ***Edition Publitronic, 1994, ISBN 2-86661-046-6.*

[BOUKHLIF-94] Aoued BOUKHLIF, **«Principes Et Fonctionnement Des Récepteurs De Télévision Monochrome Et Couleurs», ***Office Des Publications Universitaires, Alger, 1994, ISBN 2 07 2247.*

COURS.

[KARABERNOU-09] Si Mahmoud KARABERNOU, **« Méthodologie De Conception Et Langages De Description De Matériel », ***Ecole Nationale Supérieure De L'Electronique Et De Ses Applications* **(***ENSEA), Septembre 2009.*

[WEBER-00] Jacques WEBER, «**Les Circuits Logiques Programmables», ***Avril 2000***.**

[BLOTIN] T. BLOTIN, « **Le Langage De Description VHDL », *Lycée Paul-Eluard, 93206 Saint-Denis.***

[GAGNY-98] GAGNY, **« Fonctions Logiques Elémentaires Et Langages Comportementaux», *Lycée Gustave Eiffel, 1998.***

[DOUILLARD] Catherine DOUILLARD & Gérald OUVRADOU & Michel JEZEQUEL, « **ELP304/203 : Electronique Numérique, Logique Séquentielle Techniques D'intégration »,** *Telecom Bretagne***.**

THESES.

[TARIGHT-99] Y. TARIGHT, «**Contribution A L'Analyse De La Pollution Atmosphérique Chronique Ou Accidentelle : Concept De Nez Electronique Ambulatoire»,** *Thèse De Doctorat, Rouen, 19 Janvier 1999.*

[GHOZZI-03] Fahmi GHOZZI**, «Optimisation D'Une Bibliothèque De Modules Matériels De Traitement D'Images. Conception Et Test VHDL, Implémentation Sous Forme FPGA»**, *Thèse De Doctorat, L'université Bordeaux I, 2003.*

[DERRIEN-02] Steven DERRIEN, « **Étude Quantitative Des Techniques De Partitionnement De Réseaux De Processeurs Pour L'Implantation Sur Circuits FPGA »,** *Thèse De Doctorat, L'université De Rennes 1, Le 2 Décembre 2002.*

[AMERIJCKX-00] Christophe AMERIJCKX, « **Architecture D'un Réseau D'interconnexion Au Sein D'un Composant Programmable Complexe »,** *Thèse De Doctorat, L'université De Cathollque De Louvain, Louvain- La-Neuve Belgique, Août 2000.*

[DETREY-07] Jérémie DETREY, « **Arithmétiques réelles sur FPGA. Virgule fixe, virgule flottante et système logarithmique »,** *Thèse De Doctorat, L'École Normale Supérieure De Lyon, Le 15 Janvier 2007.*

[BEN ATITALLAH-07] Ahmed BEN ATITALLAH, « **Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel »,** *Thèse De Doctorat, L'université Bordeaux 1, Le 11 Juillet 2007.*

[**WOODFILL-97]** J. WOODFILL & B. VON HERZEN, "**Real-Time Stereo Vision on the PARTS Reconfigurable",** *Computer IEEE Symposium on Field-Programmable Custom Computing Machines, 1997, pp 242-250.*

[ABEL-06] Nicolas ABEL, **« Outils Et Méthodes Pour Les Architectures Reconfigurables Dynamiquement À Grain Fin. Synthèse Et Gestion**

Automatique Des Flux De Données », *Thèse De Doctorat, L'université De Cergy-Pontoise, Le 19 Mai 2006.*

[SEDCOLE-06] Nicholas Peter SEDCOLE, « **Reconfigurable Platform-Based Design in FPGAs for Video Image Processing",** *Thèse De Doctorat, L'université De London, Janvier 2006.*

[GHALI -05] M. Khemaies GHALI, « **Méthodologie De Conception Système A Base De Plateformes Reconfigurables Et Programmables »,** *Thèse De Doctorat, L'université PARIS XI ORSAY, le 01 Mars 2005.*

ARTICLES.

[MRABET-05] Hayder MRABET & Wahid BAHROUN & Zied MARRAKCHI & Habib MEHREZ, **« Intégration D'un FPGA Dans Un Système Sur Puce »,** *Setit 2005, 3rd International Conference: Sciences Of Electronic, Technologies Of Information And Telecommunications, March 27-31, 2005 – Tunisia.*

[RADUNOVIC-99] Božidar RADUNOVIC, « **An Overview of Advances in Reconfigurable Computing Systems",** *Proceedings Of The 32nd Hawaii International Conference On System Sciences – 1999 IEEE.*

[DELAHAYE -04] J. P. DELAHAYE & G. GOGNIAT & C. ROLAND & P. BOMEL, « **Software Radio And Dynamic Reconfiguration On A Dsp/Fpga Platform",** *IETR/Supelec - Campus de Rennes, Laboratoire LESTER – Université de Bretagne Sud, France.*

[DIESSEL-99] Olivier DIESSEL & Hossam ELGINDY & Martin MIDDENDORF **"Dynamic Scheduling Of Tasks On Partially Reconfigurables FPGA",** *University Of South &New South Wales- Australia, University Of Karlsruhe- Germany, September 13, 1999.*

[KARABERNOU-05] Si Mahmoud KARABERNOU & Fayçal TERRANTI**, «Real-Time FPGA Implementation Of Hough Transform Using Gradient And CORDIC Algorithm »,** *Image and Vision Computing, Volume 23, Issue 11, Pages 1009-1017, 1 October 2005.*

[GUEX-98] C. GUEX & E. MESSERLI, « **Circuits Programmables Et Langages De Conception, Une Evolution En Parallèle »,** *Janvier 1998.*

[DJUPDAL-05] Asbjørn DJUPDAL, «**Comparing Parallel Architectures from a Reconfigurable Computing Perspective »,** *9th June 2005*

[CHODOWIEC-03] Pawed CHODOWIEC & Kris GAJ, **"Very Compact FPGA Implementation Of The AES Algorithm",** *Springer, Berlin, Allemagne, 2003.*

[GUERMOUD-97] H.GUERMOUD & Y.BERVILLER & E.TISSERAND & S.WEBER, **« Architecture A Base De FPGA Reconfigurable Dynamiquement Dédiée Au Traitement D'image Sur Flot De Données»,** *Seizième Colloque GRETSI, Grenoble, 15-19 Septembre 1997.*

[MOSHER-10] Rick MOSHER & PLANO «**FPGA Prototyping to Structured ASIC Production to Reduce Cost, Risk & TTM",** *Design & Reuse, catalyst of collaborative IP based SOC design, D & R Industry Articles, March 23-24, 2010.*

[MCALLISTER-06] J. MCALLISTER & R. WOODS & R. WALKE & D. REILLY, **« Multidimensional DSP Core Synthesis For FPGA»,** *Journal of VLSI Signal Processing 43, 207–221, 2006,c_ Springer Science & Business Media, LLC.*

[POUSSIER-02] Sylvain POUSSIER & Hassan RABAH & Serge WEBER, "**SOPC-based Embedded Smart Strain Gage Sensor"** *(Eds.): FPL 2002, LNCS 2438, pp. 1131–1134, 2002. c_Springer-Verlag Berlin Heidelberg 2002*

SITES.

[Xil.com] http://www.xilinx.com**.**

[Alt.com] http://www.altera.com**.**

[Wikp.com] http://www.wikipedia.mht**.**

Résumé

La technologie d'intégration à grande échelle **VLSI** (*Very Large Scale Integration*) a bouleversée le monde des circuits intégrés. Issus de cette technologie, les réseaux logiques programmables FPGA (*Field Programmable Gate Arrays*) sont des composants **VLSI** entièrement reconfigurables, ce qui permet de les reprogrammer à volonté afin de réaliser une foule d'opérations complexes. L'intérêt d'un traitement numérique en temps réel ne cesse de croitre aussi bien en médecine qu'en spatiale ou autre. L'implantation (sur circuit) de différents algorithmes de traitement de l'information dans les appareils d'aujourd'hui devient une nécessité.
Les travaux exposés dans ce mémoire concernent la conception d'un circuit complexe en technologie FPGA. Pour maitriser la conception sur ce genre de circuit programmable ainsi que les outils de CAO qui l'accompagne nous avons choisis de réaliser un contrôleur vidéo implanté comme exemple sur un jeu vidéo. Ce dernier doit respecter des contraintes de timing très rigoureuses en rapport avec la trame VGA, ceci pour un affichage en temps réel.

Pour mettre en œuvre ce jeu vidéo et tenir en compte des contraintes de temps réel, nous avons développé des blocs décrits en langage de description matérielle de haut niveau d'abstraction **VHDL**, et nous avons établis une méthode de conception en utilisant des outils de développement de **CAO.** Nous avons réalisé deux architectures, la deuxième est dotée de quelques options supplémentaires pour l'affichage des objets. D'un autre côté ceci nous a permis de faire une étude comparative.

L'étude de cet exemple simple en apparence nous a permis d'apprendre toutes les notions nécessaires dans ce domaine pour pouvoir entamer un travail de recherche en rapport avec la conception de circuit sur FPGA et l'implantation d'algorithmes en temps réel.

Mots clés : La logique programmable, FPGA, SPARTAN 3E, NEXYS2, ADEPT, VHDL, CAO, Vidéo, VGA, Temps réel, VLSI.

Abstract

The technology of the Very Large Scale Integration **(VLSI)** has upset the world of integrated circuits. Coming from this technology, the Field Programmable Gate Arrays (FPGA) are **VLSI** components fully reconfigurables, which allows to reprogram them at will in order to achieve a crowd of complex operations. The interest of a digital treatment on real time is continuously increasing as for medicine, spatial or others. The layout (on circuit) of different algorithms for data treatment on nowadays devices becomes a necessity.

The works reported in this memorandum deal with the design of a complex circuit with **FPGA** technology. To master the design on such programmable circuit and the **CAD** tools which go with, we choose to realize a video controller implanted as an example on a video game. This one has to respect very rigorous timing constraints related with the **VGA** framework, this for a real time display.

To use this video game and with regards to real time constraints, we developed blocks described in **VHDL**: **V**HSIC **H**ardware **D**escription **L**anguage (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits) and we set up a method of design using **CAD** development. We have realized two architectures, the second is endowed with few additional options for objects framework. In the other hand, this allowed us to make a comparative study.

The study of this simple by appearance example allowed us to learn all necessary options in this field to be able to enter into a research work related with circuit design on **FPGA** and algorithms layout in real time as well.

Key words: Programmable logic, FPGA, SPARTAN 3E, NEXYS2, ADEPT, VHDL, CAD, Video, VGA, Real time, VLSI.

ملخص

قد أحدثت تكنولوجيا التكامل على نطاق كبير (VLSI)(تكامل واسع النطاق جدا) تغييرا في عالم الدارات الكهربائية الموحدة' انحدرت من هذه التكنولوجيا ،الشبكات المنطقية المبرمجة (بوابة حقل مبرمج صفيف) التي تصنف من مركبات معادة التشكيل الكامل . و التي يمكن برمجتها في كل وقت لتحقيق العديد من العمليات المعقدة. ميزة المعالجة الرقمية في الوقت الحقيقي لا تزال تتوغل في مجال الفضاء أوفي الطب أو في مجالات أخرى. أصبح تنفيذ الخوارزميات المختلفة (على الدارات) لمعالجة المعلومات في المعدات ضرورة.

تتعلق الأعمال المقدمة في هذه المذكرة بتصميم الدارة الكهربائية المعقدة في مجال تكنولوجيا (FPGA). لإتقان تصميم هذه الدارة الكهربائية القابلة للبرمجة والأدوات المستعملة للبرمجة (CAO) اخترنا تحقيق وحدة تحكم الفيديو وتنفيذها كمثال على لعبة فيديو. ويجب على هذا الأخير تلبية القيود الصارمة في التوقيت ذو الصلة مع شاشة التجهيز المرئي (VGA)، وهذا للعرض في الوقت الحقيقي.

لتنفيذ هذه اللعبة، و الأخذ في الاعتبار بقيود الوقت الحقيقي ، وضعنا مجموعة برامج مكتوبة بلغة وصف الأ جهزة على مستوى عال من التجريد (VHDL) ، وقمنا بتطوير طريقة التصميم باستخدام أدوات التطوير. قد أجرينا أبنيتين مختلفتين، تم تجهيز المركز الثاني برصيد بعض اخيتارات إضافية لعرض الأشياء على أحسن صورة. و من ناحية أخرى' قد سمحت لنا بتقديم دراسة مقارنة.

قد أجاز لنا دراسة هذا المثال البسيط تعلم كل المفاهيم المطلوبة في هذا المجال من أجل البدء في العمل البحثي في تصميم الدارات الكهربائية على (FPGA) وتنفيذ مختلف الخوارزميات في الوقت الحقيقي.

كلمات البحث : منطق البرمجة ، مركب FPGA، سبارتن 3E، نكسيس2، أدابت، لغة وصف الأجهزة VHDL، أدوات البرمجة CAO، فيديو، شاشة VGA ، الوقت الحقيقي، مركب VLSI