IUT de l'Indre
Département Génie Electrique et Informatique Industrielle

Module ENSL1 : Composants logiques reconfigurables

Cours

Eric PERONNIN

Chateauroux, le 4 octobre 2012

Table des matières

Chapitre 1

Introduction : synthèse de systèmes logiques en circuit

1 Les composants logiques reconfigurables en bref ...

Il s'agit de composants mettant à la disposition du concepteur un ensemble de portes aux interconnexions reconfigurables et permettant la mise en oeuvre :

- de sommes de produits de termes (des fonctions ET de plusieurs entrées dont on réalise la somme avec une fonction OU),
- de mémoires de base au niveau du bit, c'est à dire pour stocker un unique signal scalaire (bascules D, RS, T, JK),
- de mémoires vectorielles (ROM, RAM, RAM double accès),
- de systèmes autorisant la gestion des horloges dans le circuit (PLL, réseau d'interconnexion dédié),
- de tout un ensemble de blocs d'entrées/sorties configurables pour s'adapter à l'environnement d'utilisation (signaux TTL, CMOS, LVTTL, LVCMOS, PCI ...).

2 Pourquoi utiliser des composants logiques reconfigurables ?

La création d'un nouveau circuit intégré assurant un ensemble de fonctions logiques est une opération très coûteuse en temps et en moyens. On estime que le développement d'un ASIC [1] Full Custom (un circuit intégré dont la structure est gravée et figée) nécessite en moyenne 18 mois et que chaque remise en cause du design (à cause de bug par exemple) peut entraîner un retard de l'ordre de 3 à 6 mois pour réaliser une nouvelle puce testable.

De fait, ce type de développement était historiquement et reste encore aujourd'hui reservé à des projets avec des objectifs de grandes séries.

Par ailleurs, la complexité des puces de fortes densités telles que des processeurs ont apporté leur lot de contraintes supplémentaires. Le nombre d'éléments logiques impliqués dans de tels projets a eu rapidement pour conséquence de rendre les simulations fonctionnelles extrêmement longues et difficiles, même avec les moyens informatiques d'aujourd'hui.

C'est pour ces deux types de production que des composants logiques programmables de fortes densités, c'est à dire abritant plusieurs millions de portes logiques configurables et interconnectables à volonté, ont fait leur apparition avec les avantages suivants :

- pour le développement de circuits de fortes densités, il est possible de réaliser une simulation en circuit des propriétés fonctionnelles d'un design. Le concepteur d'un microprocesseur ayant décrit son projet dans un langage de haut niveau avec un outil de développement intégré transfert l'ensemble de son design dans des composants logiques programmables de fortes densités et peut ainsi apprécier très rapidement le résultat de son travail avec une vitesse de traitement proche de la vitesse qu'atteindra

1. Application Specific Integrated Circuit

la puce définitive. En cas d'erreur, il suffit de modifier le design (du code en général), de recompiler le projet et, après une étape de placement/routage, de charger de nouveau le projet dans les composants programmables. De 4 mois pour la fabrication d'une puce mise à jour dans le cas des ASIC Full Custom, on tombe à quelques heures de retraitement avant un nouveau test avec des composants logiques reconfigurables haute densité. Lorsque le projet semble définitif, il est alors possible de lancer la fabrication en série d'une puce gravée, figée, à partir d'un masque (on reprend le processus final de développement d'un ASIC) ou, comme c'est le cas aujourd'hui, de commander la fabrication de la puce dédiée au fondeur du composant programmable (technologie HardCopy [2] d'Altera par exemple).

- dans le cas de composants destinés à de petites séries, on ne grave pas de circuit spécifique à l'issue de la phase de conception. C'est un composant logique programmable, associé à une mémoire de configuration dans le cas des composants programmables reposant sur une technologie volatile, qui sera fourni à l'utilisateur final. Le gain de temps excède alors d'une année le développement d'une puce ASIC Full Custom et les coûts de production finaux sont comparativement dérisoires.

3 Outils de développement

Les premiers composants logiques programmables (PLD pour Programmable Logic Devices) contenaient un nombre de portes limité (de l'ordre de 4000) et disposaient grossièrement au plus d'une vingtaine d'entrées et d'une douzaine de sortie. Il était possible de calculer n'importe quelle équation combinatoire à partir de ces entrées et de réaliser des systèmes séquentiels simples.

De part leur faible densité, il était aisé de les exploiter totalement avec des environnements de développement rustiques basés sur des langages dits de première génération tels que PALASM (AMD), AHDL (Altera), Abel HDL (Data I/O) ...

Avec l'apparition des premiers CPLD et FPGA, les concepteurs ont rapidement exprimé le besoin d'outils de plus haut niveau, capable d'offrir un langage autorisant la synthèse de dispositifs complexes et gourmands en nombre de portes avec une approche hiérarchique. A l'instar de Verilog HDL son concurrent, le langage VHDL (Very high speed integrated circuit Hardware Description Language) est apparu comme une solution intéressante.

Les différents fondeurs de composants ont rapidement proposé des outils de développement intégrés permettant de prendre en charge la conception complète d'un circuit : du design à la programmation finale du chip en passant par diverses étapes de simulation.

Ainsi, et puisqu'il autorise des descriptions de modules en langage VHDL, ce cours fera régulièrement référence au logiciel Quartus II d'Altera.

2. voir http ://www.altera.com/products/devices/

Chapitre 2

Les composants logiques reconfigurables

1 Introduction

Il en existe deux grandes familles :

- les FPGA : Field Programmable Gate Array (Réseaux de champs de portes programmables)
- les CPLD : Complex Programmable Logic Device

Pour faire simple, les FPGA reposent sur une structure matricielle constituée de cellules élémentaires assurant des calculs combinatoires sur peu d'entrées (moins de 10) avec la possibilité d'en mémoriser le résultat dans une bascule. Les CPLD, à l'opposé, une architecture distribuée constitué d'énormes blocs de calculs combinatoires capables de traiter de très nombreuses entrées (parfois toutes les entrées du circuit)

Dans 90% des applications, les FPGA peuvent être utilisés. Le champs d'apptitude des CPLD étant beaucoup plus restreint et adapté principalement aux systèmes gourmands en calculs combinatoires.

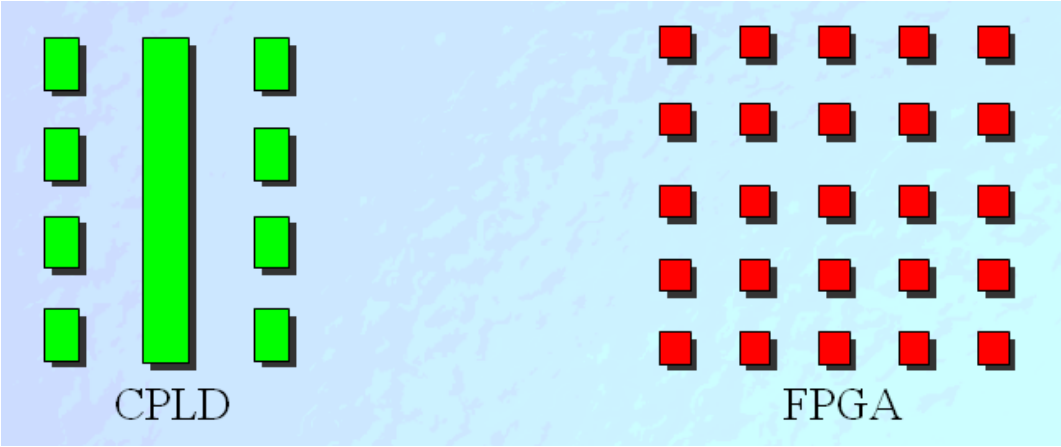

FIGURE 2.1 – Différences de structure des FPGA et CPLD

2 Les CPLDs par l'exemple : la famille XC9500 de Xilinx

2.1 Quelques caractéristiques de la famille XC9500XL

C'est une liste non exhaustive (voir la datasheet Xilinx pour en savoir plus sur www.xilinx.com) :

- Tension d'alimentation de 3.3v pour les XC9500XL.
- Compatible 5v, 3.3v et 2.5v en entrée ; 3.3v et 2.5v en sorties.
- Fréquence maximale de fonctionnement jusqu'à 208 MHz.
- Jusqu'à 6400 portes (équivalent) et 288 macro-cellules.
- Technologie Flash non volatile (10000 cycles).

2.2 Structure interne des CPLDs XC9500XL

Elle correspond au schéma de la figure 2.2. Les différents éléments sont détaillés dans la suite.

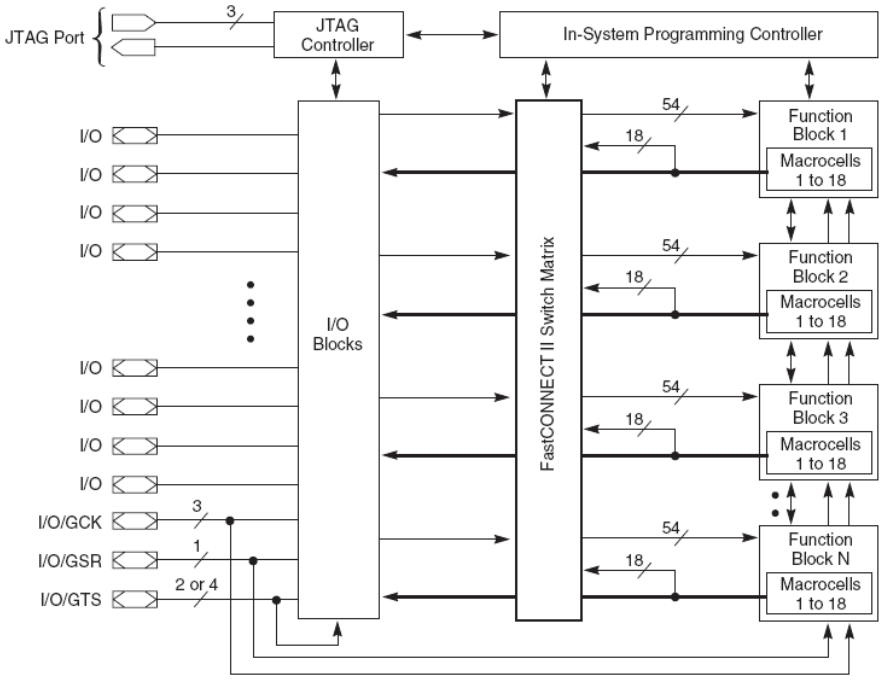

FIGURE 2.2 – Structure interne des XC9500XL

2.3 Bloc de fonctions (Function Block) et macro-cellules (Macrocell)

Les XC9500 exploitent des blocs fonctionnels capables de traiter un grand nombre d'entrée à la fois pour des calculs combinatoires et la synthèse de machines séquentielles complexes. C'est en étudiant la macro-cellule d'un CPLD qu'on prend la mesure de la technologie de ces CPLDs (à comparer avec ce qui sera vu plus tard pour les FPGAs).

Comme le montre la figure 2.3, un bloc fonctionnel est un regroupement de 18 macro-cellules.

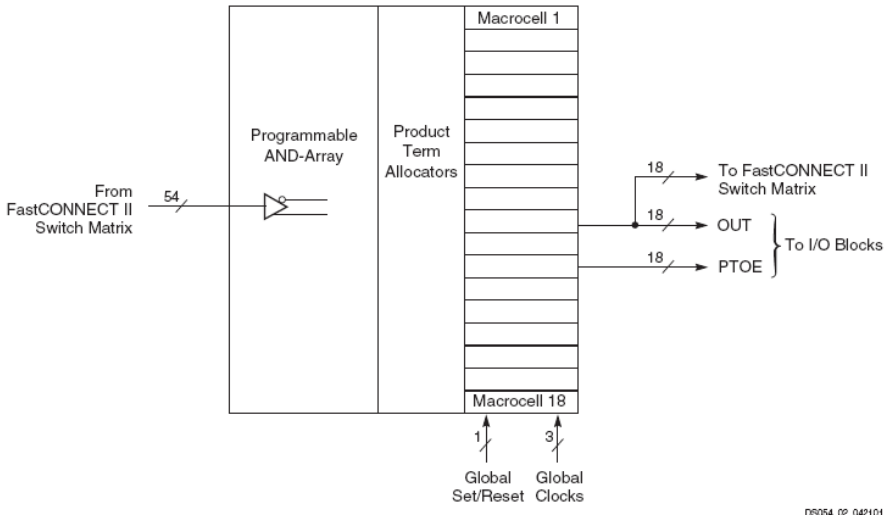

FIGURE 2.3 – Function Block

En amont des macro-cellules, le bloc fonctionnel fait apparaître deux premiers éléments :

- le programmable AND Array assurant la complémentation (pour disposer d'une forme complémentée de chacune des entrées) et l'amplification (buffering) des 54 entrées (La structure de ces 18 macro-

cellules apparaît à la figure 2.5 et mérite quelques explications) puis le calcul des différents produits qui interviennent dans les équations combinatoires à synthétiser (90 lignes de ET de 108 entrées ; les 54 non complémentées et leurs homologues complémentées).

- les Product Terme Allocators (détail figure 2.4) dirigent les produits de termes vers les macrocells en fonction des besoins, accroîssent le potentiel en sommes de termes d'une Macrocell. Dans les faits, ils apportent 5 produits à chaque macro-cellule et assurent le calcul d'un terme d'extension sous la forme d'une somme de produits pour la macro-cellule qu'ils alimentent en signaux.
  Note : le terme d'extension recevant des signaux provenant des macro-cellules inférieure et supérieure, et leur fournissant également des signaux, il est ainsi possible de calculer des sommes de produits faisant intervenir de très nombreux produits. Ce mécanisme de communication entre macro-cellules permet par ailleurs d'optimiser les calculs arithmétiques mettant en jeu des retenues.
- 18 macro-cellules (macrocells), la macro-cellule constituant l'élément de base des CPLDs.
- PTOE (Product Term Output Enable) est un signal associé à une sortie, autorisant ou non, la sortie à être véhiculée sur une broche du composant (PTOE pilote en fait une porte 3 états par laquelle la sortie associée de la macro-cellule transite. Voir la suite).
- Global Set/Reset sont des signaux issus de 2 broches dédiées du composants permettant la remise à 0 ou la mise à 1 asynchrones des bascules de sortie des macro-cellules.
- Global Clocks est une connexion directe d'une broche dédiée du composant vers les entrées d'horloge des bascules des macro-cellules (optimisation du trajet).

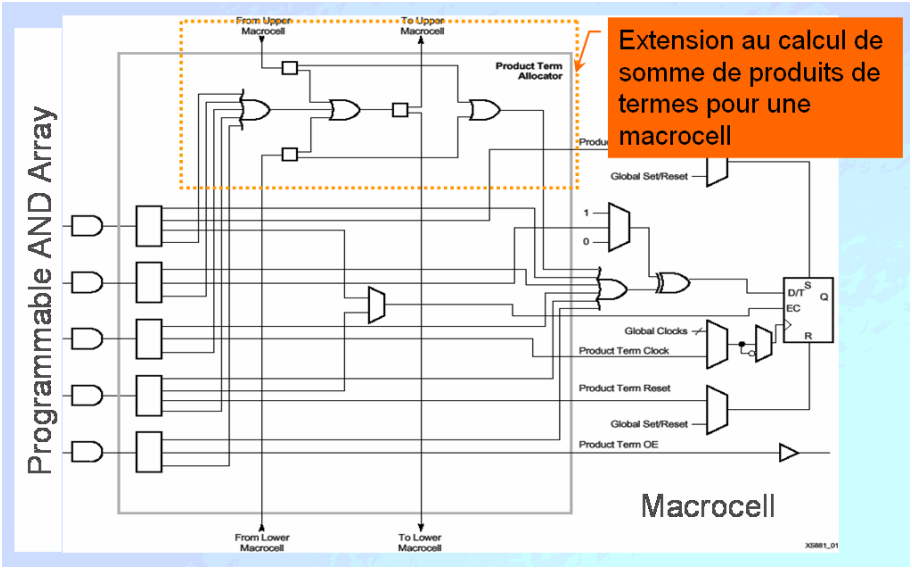

FIGURE 2.4 – Product Terme Allocator

L'étage de sortie de la macro-cellule (figure 2.5) fait clairement apparaître :

- La porte OU en entrée pour concrêtiser le calcul de la somme de produits.
- Un OU EXCLUSIF utile dans le cadre du calcul des bits de retenus dans les calculs arithmétiques binaires.
- Une bascule pour mémoriser le résultat et rendre possible la réalisation de systèmes séquentiels.
- Un multiplexeur de sortie permettant de choisir entre le signal purement combinatoire ou le signal mémorisé à la sortie de la macro-cellule.

Note : la sortie est dirigée vers le bloc d'entrée/sortie d'une part et vers la matrice d'interconnexion qui fournit les blocs fonctionnels d'autre part.

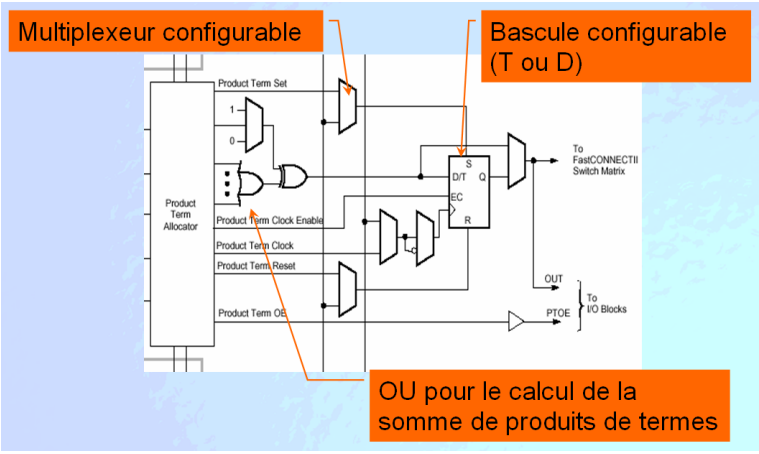

FIGURE 2.5 – Macrocell

2.4 Blocs d'entrées/sorties

Ils assurent la connexion avec l'extérieur du composant en tentant d'offrir une compabilité la plus large possible en termes de signaux. Ainsi, dans le cadre d'une configuration en sortie, la sortie repose sur un amplificateur 3 états permettant de mettre son signal dans un état de haute impédance suivant une consigne fournie par PTOE, GTS1, GTS2 ou encore par une information provenant d'une autre macro-cellule que celle à l'origine du signal de sortie. Lorsque la broche est configurée en entrée, le signal lui parvenant passe par un buffer d'amplification. Sa structure est présentée figure 2.6.

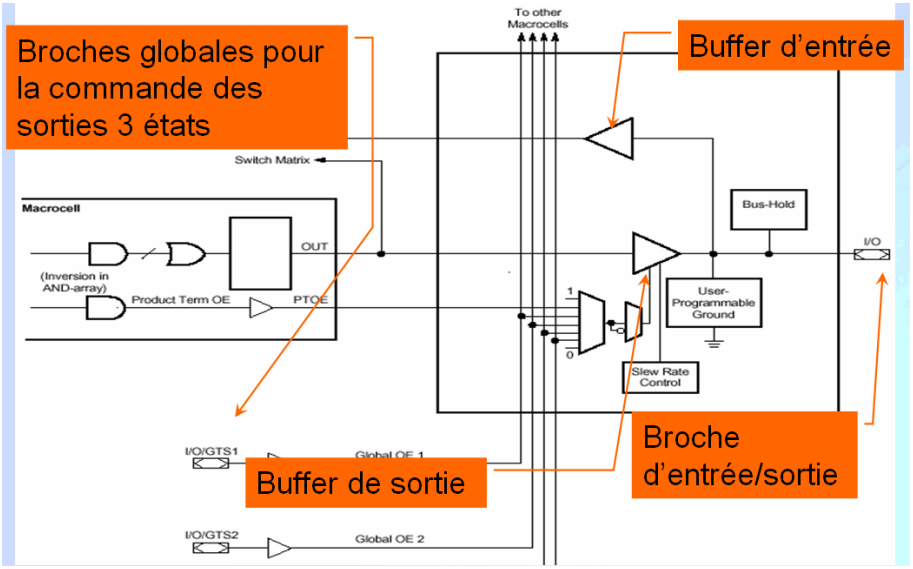

FIGURE 2.6 – I/O Block

Note :

- Les éléments USER PROGRAMMABLE GROUND et BUS HOLD permettent d'assurer des tirages à la masse ou au VCC si besoin est.
- Pour les sorties, le slew rate (temps de montée) du buffer de sortie peut être contrôlé.

Les compatibilités de signaux à partir de ces blocs d'entrées/sorties sont les suivantes :

- Buffer d'entrée (XC9500XL)
  - Compatible avec les signaux 5v CMOS, 5v TTL, 3.3v CMOS et 2.5v CMOS
  - Référence 3.3v interne pour la stabilité
  - Hystérésis de 50mV pour l'immunité au bruit
- Buffer de sortie (XC9500XL) : la tension de sortie est fixée par la broche dédiée du CPLD : VCCIO

- 3,3v : compatibilité avec les circuits 3.3v CMOS et 5v TTL
- 2,5v : compatibilité avec les circuits 2.5v CMOS

2.5 Configuration du circuit : chargement d'un design

Cette configuration exploite le port JTAG qui est un port de communication série reposant sur un protocole standard.

Elle peut être effectuée à partir d'un PC disposant de l'interface JTAG ou à l'aide d'un programmateur de composants.

Note importante : l'information de configuration dans les CPLDs XC9500 (comme dans de nombreux autres CPLDs) est stockée dans une mémoire de type FLASH si bien qu'elle est conservée en cas de coupure de l'alimentation.

3 Les FPGAs : illustration avec la famille Cyclone II Altera [1]

3.1 Introduction

Les fabricants de FPGAs sont aujourd'hui très nombreux sur le marché. Ils demeurent cependant deux leaders historiques dont la suprématie est toujours réelle aujourd'hui : Xilinx (série Virtex en haut de la gamme) et Altera (série Stratix pour le sommet de la production).

Les FPGAs ont une structure matricielle dont les éléments de base permettent des traitements logiques combinatoires et/ou séquentiels opérant sur peu de données binaires. Typiquement, l'élément de base, appelé CLB pour Complex Logic Block chez Xilinx ou LE pour Logic Element chez Altera, permet de traiter des équations à 4 entrées auxquelles peuvent s'ajouter des signaux de calculs en chaîne tels que le signal de retenu apparaîssant lors d'une addition binaire. En limitant le nombre d'entrées traitées dans un CLB ou LE, les concepteurs ont visé une bonne optimisation du taux d'utilisation des portes présentes dans le composant lors des calculs combinatoires (à l'opposé des CPLD conçus pour des calculs mettant en jeu beaucoup d'entrées).

Parce qu'ils disposent d'un programme de dotation universitaire permettant d'être équipé totalement gratuitement, il y a quelques années, j'avais fait le choix de travailler avec des FPGA Xilinx. La gamme utilisée jusqu'ici en travaux pratiques, à savoir la famille XC4000XL, n'existant plus au catalogue Xilinx et n'étant plus gérée par les dernières versions du logiciel ISE Foundation Xilinx dans sa version Web gratuite, j'ai choisi d'utiliser la technologie Altera pour la promotion 2008-2009. De fait, les 400 CLB du XC4010XL que nous utilisions par le passé ont été remplacés par les 35000 LE du Cyclone II EP2C35F672C6 qui est un FPGA mature de moyenne densité au sein de la gamme Altera 2008.

3.2 Caractéristiques principales de la famille Cyclone II

Les Cyclone II sont venus compléter la gamme Cyclone fin 2004.

Leurs principales caractéristiques sont :

- Une gravure en 90 nm. C'est un paramètre important car plus la gravure est fine et plus l'élément de base qu'est le transistor est petit et voit sa consommation et donc ses dégagements en chaleur diminués. Note : graver plus fin permet également de placer davantage de portes dans le FPGA à surfaces égales.
- Une architecture haute densité avec entre 4608 et 68416 LE (Eléments logiques).
- Ils intègrent jusqu'à 1.1 Mbits de RAM (M4K Blocks) avec :
  - une largeur de bus de données configurable (x1, x2, x4, x8, x16, x32 et x36),
  - un véritable mode double accès,
  - une vitesse de fonctionnement pouvant atteindre 260 MHz.
- Ils comprennent une zone où sont gravés dès la fabrication jusqu'à 150 Multipliers 18x18 (Embeded Multipliers).

1. Consulter le Cyclone II Device Handbook volume 1 pour plus d'informations

- Et ils possèdent des blocs d'entrées/sorties avancées (IOE : Input/Output Element) avec :
  - des entrées/sorties différentielles (LVDS, LVPECL ...),
  - des entrées/sorties simples (3.3v, 2.5v, 1.8v et 1.5v LVCMOS, 3.3v, 2.5v et 1.8v LVTTL ...),
  - une compatibilité avec les signaux PCI et PCI Express.
- Il y a une circuiterie dédiée aux horloges (PLL) :
  - 2 à 4 PLL pour multiplier ou diviser les fréquences d'horloge entrantes, les retarder ...
  - jusqu'à 16 lignes d'horloges en interne
  - une gestion jusqu'à la fréquence maximale de 402.5 MHz
- Enfin, un dispositif de configuration avec :
  - un mode rapide pour une configuration en moins de 100 ms
  - mode série ou JTAG possible avec des mémoires de configuration série de bas coût

Comme de nombreuses autres familles Altera, de nombreux modules de propriétés intellectuelles sont disponibles (processeur, DSP, interfaces, communication ...) .

Par ailleurs, le logiciel SOPC Builder permet de construire un système complet sur une puce Cyclone II en quelques clics, intégrant processeurs, interfaces diverses, mémoires de programmes, de données, des contrôleurs d'écran ...

3.3 Vu de l'intérieur

Les différents FPGA du marché (Altera, Xilinx, Atmel ...) ont des structures internes assez semblables. Il n'est pas limitatif d'un point de vue conceptuel de ne s'intéresser qu'à la famille Cyclone II d'Altera.

3.3.1 Vision interne globale

Elle est représentée figure 2.7.

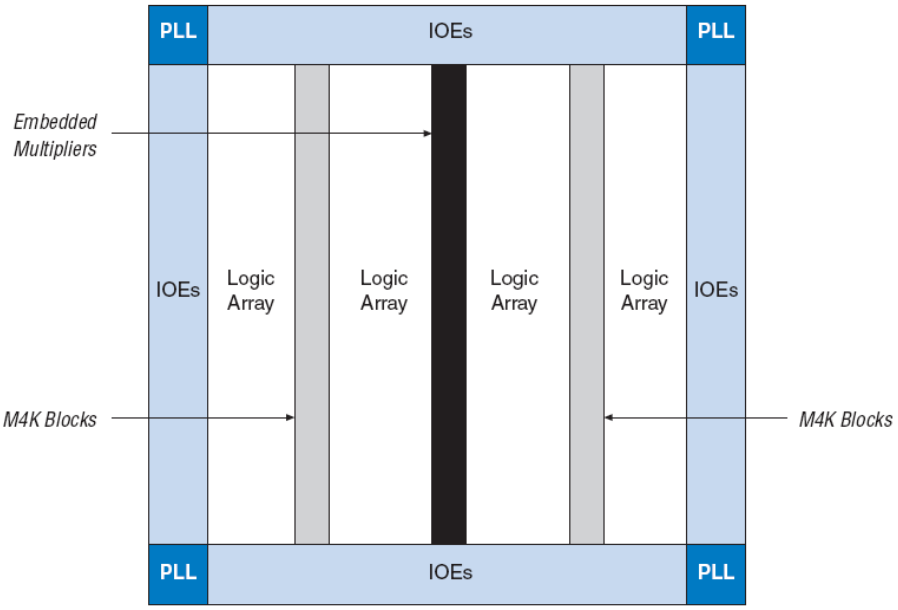

FIGURE 2.7 – Structure interne d'un Cyclone II

Les Logic Array sont constitués de plusieurs blocs de réseaux logiques (LAB : Logic Array Block). C'est à l'intérieur de ces blocs que les systèmes combinatoires et séquentiels réalisés par le concepteur se matérialisent au niveau de éléments logiques (LE : Logic Element)

3.3.2 Elément logique (LE)

C'est le plus petit élément de logique dans le Cyclone II. Sa structure apparaît sur la figure 2.8.

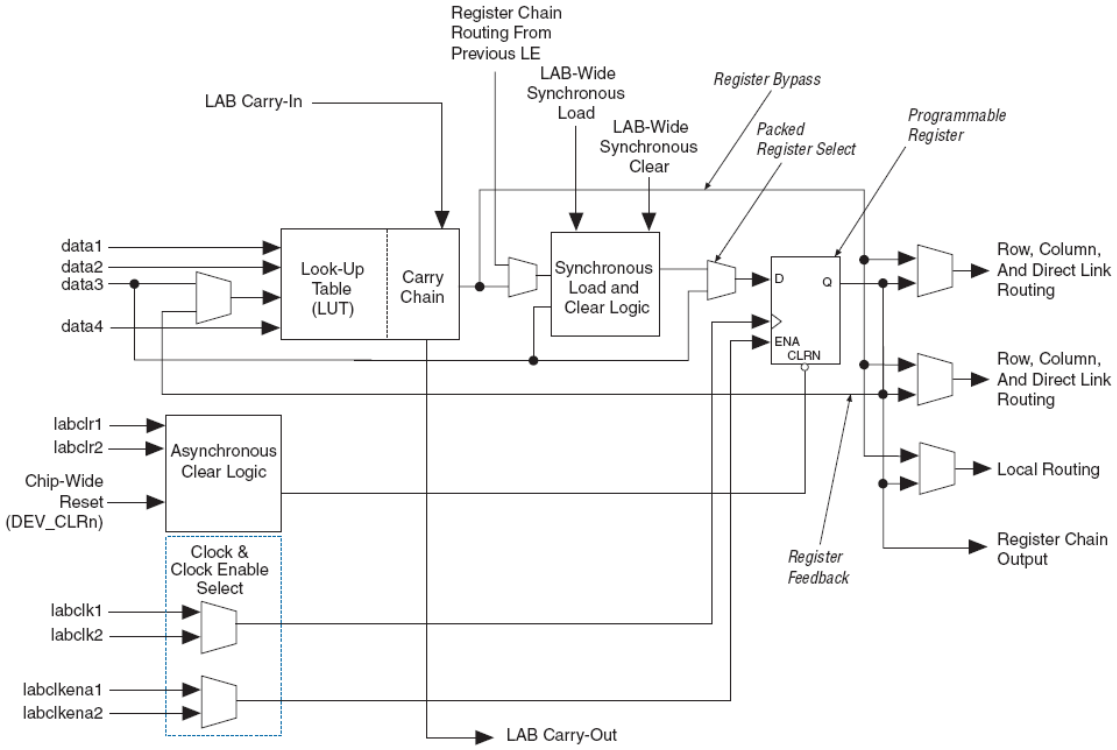

FIGURE 2.8 – Structure d'un Logic Element

Cette cellule reconfigurable (on parle davantage de reconfiguration que de programmation dans le cas d'un composant logique) contient divers éléments :

- une LUT (Look up table) à 4 entrées permettant le calcul de n'importe quelle fonction de 4 entrées,
- 1 registre de sortie programmable,
- Parce que les Logic Element sont regroupés par 16 dans des Logic Array Blocks, ils disposent de plus :
  - d'une entrée spécifique de retenue dédiée pour les calculs chaînées (Carry Chain) provenant du LE précédent dans le même LAB,
  - 1 signal de Carry de sortie destiné au LE suivant dans le même LAB.

Note :

La LUT du LE peut être utilisée suivant deux modes :

- Combinatoire : on dispose alors d'une LUT à 4 entrées.
- Arithmétique : la LUT se scinde en 2 LUT à 3 entrées, l'une fournissant le signal de sortie qui pourra être mémorisé dans le registre de sortie ; l'autre assurant de calcul d'une retenue, la Carry de sortie.

3.3.3 Les blocs de réseaux logiques (LAB)

Un bloc de réseaux logiques contient :

- 16 Eléments Logiques (LE).
- des signaux dédiés pour optimiser les calculs entre LE (signal de carry par exemple).
- Un réseau d'interconnexion local pour la communication entre LE du même LAB (voir figure 2.9).
- Un accès direct aux éléments adjacents du LAB dans la structure du FPGA (voir figure 2.10) tels que :
  - un autre LAB,
  - un bloc mémoire pour les LAB adjacents aux zones mémoires,
  - un signal d'horloge d'une PLL,
  - un multiplier,

- un IOE (Elément d'entrée/sortie),
- un accès aux réseaux d'interconnexions lignes/colonnes pour atteindre n'importe quel point du composant.

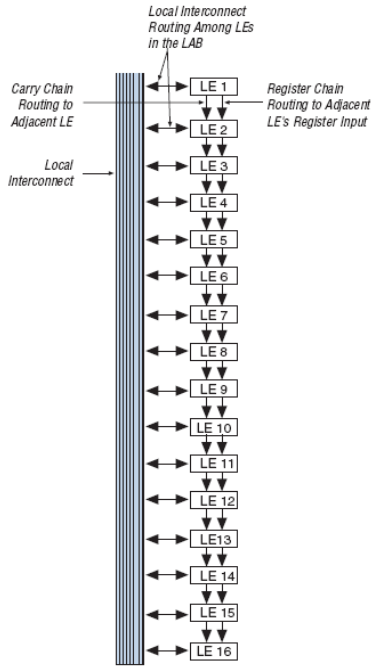

FIGURE 2.9 – Connexions chaînées entre LEs dans un LAB

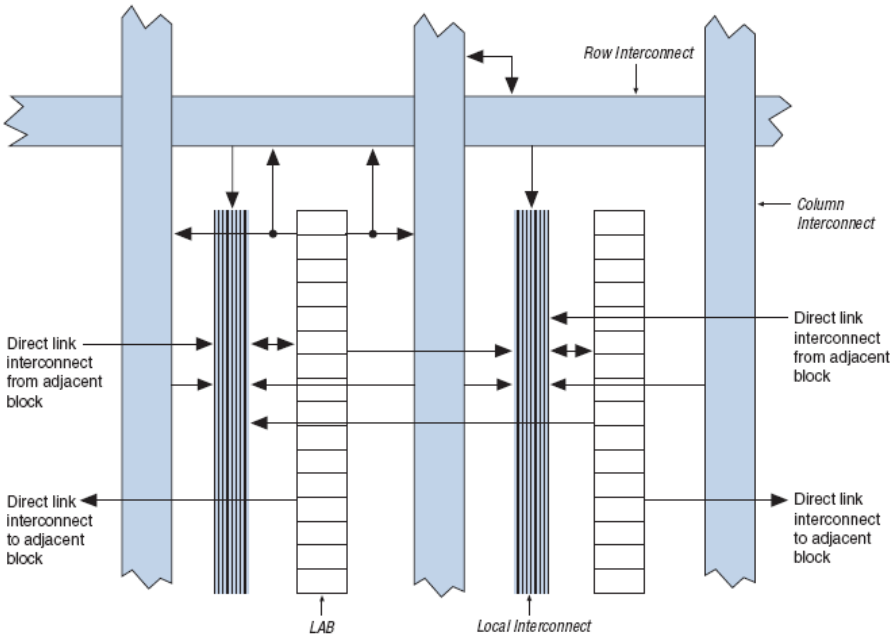

FIGURE 2.10 – Connexion directe au niveau d'un LAB

3.3.4 Interconnexions longues distances

Les deux sections précédentes ont fait apparaître deux types d'interconnexions : entre LEs d'une part et entre éléments de natures variées adjacents aux LABs d'autre part (mémoire, autre LAB, IOE, multiplier).

Compte tenu de la structure très étendue du circuit et du nombre de LABs importants (jusqu'à 4000), 2 autres voies d'interconnexions sont possibles et visent à permettre des liaisons rapides entre LABs éloignés (transit en colonne de 4 en 4 LABs ou 24 en 24 LABs ; en ligne de 4 en 4 LABs ou de 16 en 16).

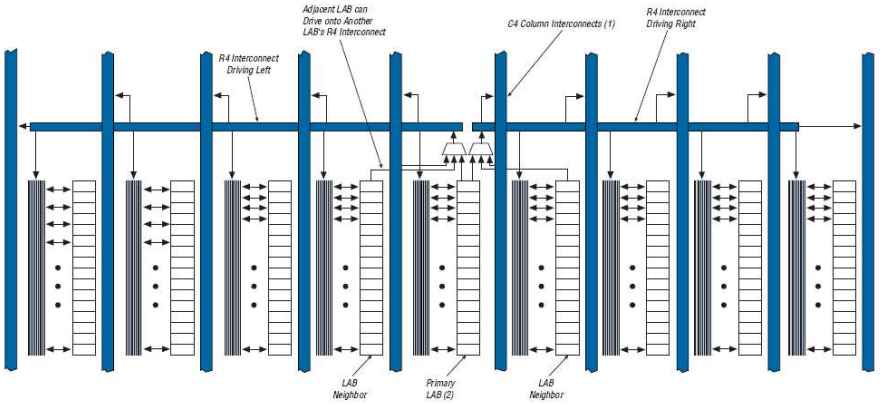

FIGURE 2.11 – R4 interconnect

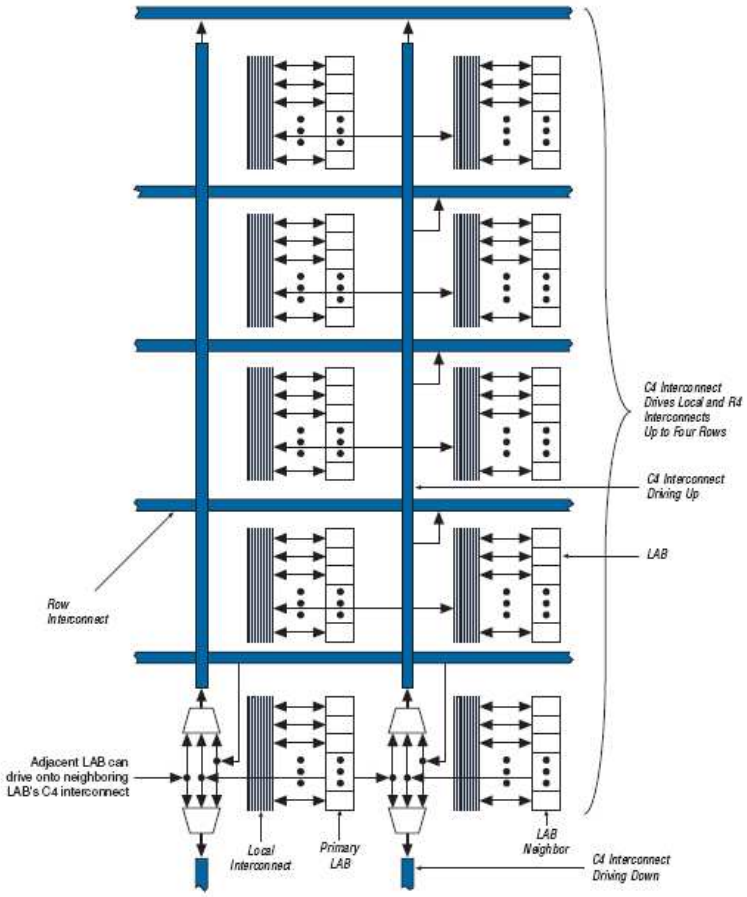

FIGURE 2.12 – C4 interconnect

3.3.5 Gestion des signaux d'horloge

C'est un point crucial à l'intérieur des FPGAs (et même à l'intérieur des circuits numériques rapides en général). En effet, pour que les phénomènes dans un circuit numérique restent parfaitement synchrones, la structure interne du composant doit être rigoureusement pensée en terme de temps de propagation des signaux d'horloge.

Pour atteindre les objectifs en la matière, la famille Cyclone II exploite différentes techniques :

- Les signaux d'horloge ne transitent pas par le réseau d'interconnexions des autres signaux mais disposent d'un réseau dit "d'horloges globales" pour assurer des liaisons à la durée optimisée (ce réseau peut compter jusqu'à 16 horloges différentes dans la famille Cyclone II).

- Dans de nombreuses situations, l'existence de signaux d'horloge de fréquence élevée à l'extérieur du composant peut présenter des difficultés de conception (au niveau du circuit imprimé particulièrement). Dans d'autres cas, le concepteur peut avoir besoin de plusieurs fréquences d'horloge différentes, décalées dans le temps, multiples les unes des autres ... Pour ces différentes raisons, la plupart des FPGAs d'aujourd'hui intègrent des boucles à verrouillage de phase (PLL : Phase Locked Loop ; voir programme d'électronique de seconde année). Ces PLLs (4 au maximum dans la famille Cyclone II) permettent donc de fabriquer des horloges répondant aux différents besoins du concepteur sur la base d'une horloge externe de fréquence modérée.

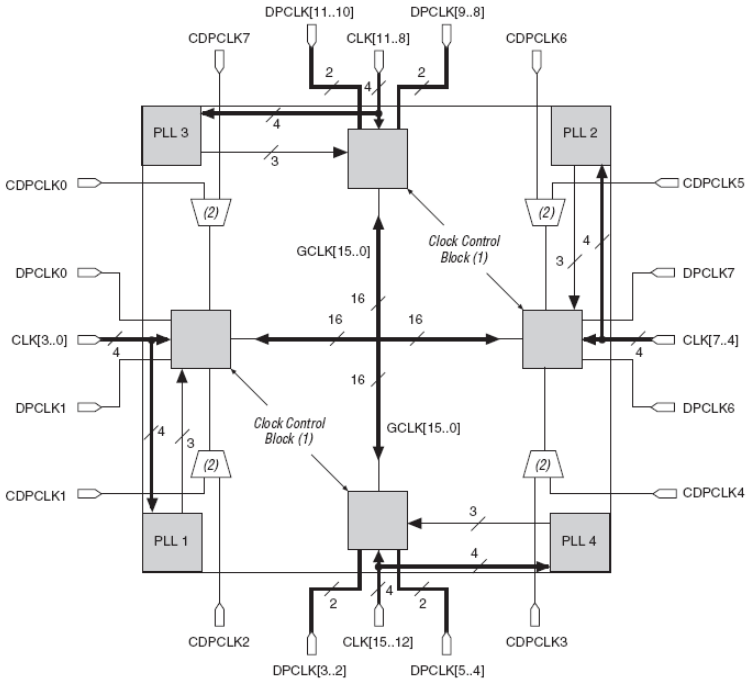

FIGURE 2.13 – Global Network Clock - PLL

3.3.6 Bloc d'entrée/sortie (IOE : Input Output Element)

Avec l'accroissement constant du nombre de standards d'entrées/sorties en électronique numérique, la conception d'un FPGA a progressivement nécessité de faire apparaître des blocs dédiés capables d'interfaçages s'adaptant à une grande diversité de situation.

La frange supérieure des composants de la famille Cyclone II dispose donc de nombreux blocs d'entrées/-sorties (I/O Banks) répartis à la périphérie du composant comme le montre la figure 2.14.

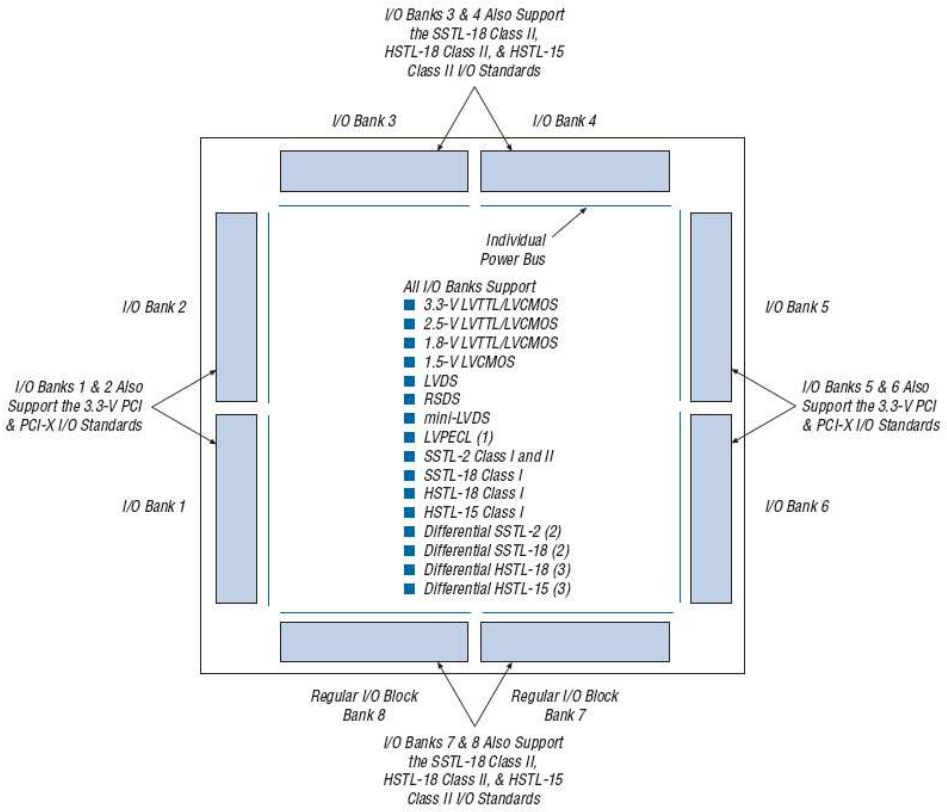

FIGURE 2.14 – Répartition des IOE (Input Output Elements) dans les I/O Banks - Signaux pris en charge

En bref, la famille Cyclone II peut s'interfacer avec des circuits logiques :

1. LVTTL et LVCMOS : Interfaçage avec des circuits logiques d'usage général, fonctionnant à des fréquences en deça de 100MHz.

- SSTL : Standard mis en place pour la mémoire SDRAM DDR (Double Data Rate).
- HSTL : Signaux des mémoires QDR [2] SRAM (Quad Data Rate).
- LVDS (Low Voltage Differential Signaling) : Signaux différentiels (ils garantissent une plus grande immunité au bruit) pour des communications à fort débit (jusqu'à 805 Mbps) et faible EMI (émissions électromagnétiques)
- LVPECL (Low Voltage Positive Emitter Coupled Logic) : Signaux différentiels à haute immunité au bruit utilisés en video, télécom, distribution d'horloge.
- PCI et PCI Express Bus locaux des PC utilisés pour la connexion de carte d'extension (Video ...).

3.3.7 Fonctionnalités de traitement du signal : multipliers embarqués

Pour les applications gourmandes en traitement du signal, la famille Cyclone II intègre de nombreux multipliers directement gravés sur la puce afin de ne pas avoir à grèver le potentiel de LE (la réalisation d'un multiplier est exigeante en nombre de portes).

Note : la télévision numérique, le son et l'image en général, sont des cibles privilègiées où les traitements nécessitent des multiplications massives à des cadences assez importantes. Cela justifie l'existence de telles structures dans les FPGA.

A l'intérieur des FPGAs Cyclone II, les multipliers opérent sur des mots de 18 bits avec un résultat sur 36 bits et sont regroupés dans des blocks placés en colonnes dans le FPGA qui peut comporter jusqu'à trois colonnes de ces multipliers (cf. figure 2.15).

2. Quad Data Rate : mémoire à l'accès en écriture sur fonts montants et descendants de l'horloge d'écriture et accessible en lecture sur les fronts montants et descendants d'une horloge de lecture.

Note : alors qu'un processeur classique pourra effectuer jusqu'à 8 multiplications par cycle d'horloge, il est possible dans un temps semblable d'en réaliser de 4 à 20 fois plus dans un FPGA.

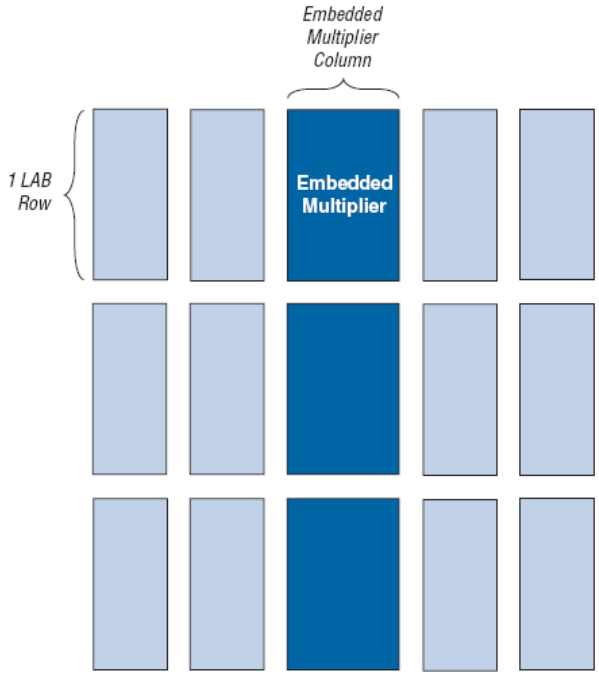

FIGURE 2.15 – Structure en colonne des multipliers au milieu de LABs

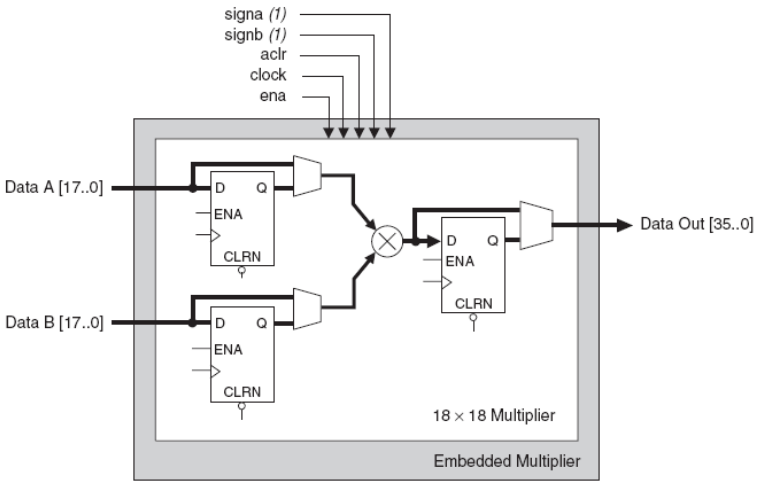

FIGURE 2.16 – Structure interne d'un multiplier en mode 18 bits

Note : pour être plus précis, il existe deux modes de configurations des multipliers :

- un premier mode où les multipliers fonctionnent en 18 bits (cf. figure 2.16).
- un second mode permettant d'obtenir deux multipliers 9 bits x 9 bits en lieu et place d'un modèle 18 bits x 18 bits.

3.3.8 Dispositif de configuration

Contrairement au CPLD vu précédemment, les FPGAs Cyclone II conservent leur configuration en interne avec une mémoire vive statique (SRAM). Par conséquent, cette configuration disparaît à chaque coupure de la tension d'alimentation.

Durant les phases de tests ainsi qu'une fois le design validée, il est nécessaire de pouvoir effectuer la configuration du FPGA pour transférer le design conçu avec le logiciel de développement Quartus II.

Dans le cas de la famille Cyclone II, Altera a mis en place 3 modes de configuration :

- deux sont à réserver aux phases de développement :
  - le mode Passive Serial Configuration rend le FPGA passif (esclave) et permet à un élément actif (le maître) de configurer les différents éléments internes du composant.
  - le second mode exploite le bus JTAG qui, outre la possibilité de configurer le composant, apporte au concepteur de nombreux autres avantages au cours des phases de développement. En particulier, il permet aussi d'observer les signaux internes du FPGA durant son fonctionnement.

  Note : le contrôle total du transfert de l'information de configuration nécessite un élément actif sophistiqué. C'est ce qui explique la présence d'un troisième mode dont l'exploitation est beaucoup moins coûteuse.
- un mode pour la configuration du FPGA au design achevé :
  - c'est le mode Active Serial Configuration. Le FPGA dispose d'une logique de commande interne lui permettant de contrôler le chargement de sa configuration interne à partir d'une mémoire passive extérieure peu coûteuse à fabriquer.

Il est important de noter qu'il est possible de configurer plusieurs FPGAs à partir d'une seule mémoire de configuration. De la même façon, bus JTAG et configuration passive (voir figure 2.17) permettent de prendre le contrôle de plusieurs FPGAs reliés ensemble et disposés sur un même PCB.

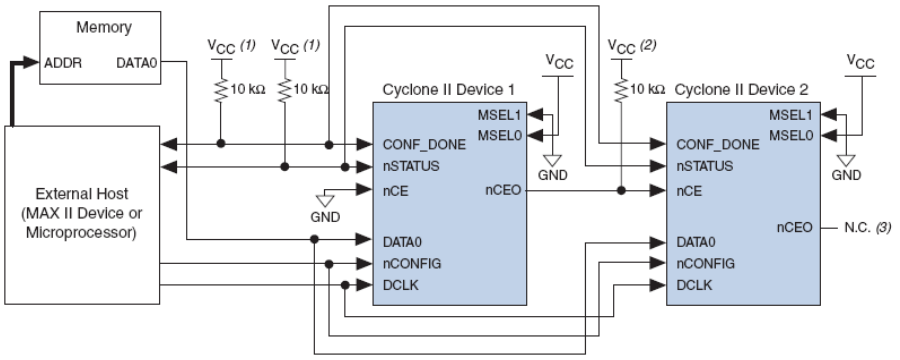

FIGURE 2.17 – Configuration passive de plusieurs Cyclone II chaînés

Chapitre 3

La plateforme de développement Quartus II d'Altera

En bref... Même s'il existe des environnements multi-fondeurs (Active-HDL, Cadence ...), les outils de fabricants tels que ceux de Xilinx ou Altera constituent une excellente base de travail. Ce chapitre propose une illustration succinte de l'outil de conception Quartus II d'Altera et commence donc par une présentation du cycle de développement d'un composant logique reconfigurable.

1 Design Flow

Pour mener à bien un projet dans sa globalité, la plateforme Quartus II intègre de très nombreux outils et permet de prendre en charge le cycle (cf. figure) de développement complet d'un PLD (CPLD ou FPGA) où se succèdent les étapes suivantes :

- Saisie du projet sous différentes formes :
  - Saisie de schémas : elle utilise des blocs logiques de base (portes logiques, modules arithmétiques, éléments d'interfaçage, blocs définis par l'utilisateur en langage VHDL ou dans un autre mode de représentation, entrées/sorties...).
  - Représentation par une machine d'état.
  - Code écrit en VHDL (dans le cadre du cours ENSL1) mais aussi en Verilog HDL, AHDL ...
- Simulation fonctionnelle (RTL [1]) : elle permet de vérifier la validité de la conception à un niveau fonctionnel sans faire intervenir les temps de propagation dans le circuit.
- Synthèse : c'est l'étape qui minimise et optimise la netliste RTL. A l'issue de la synthèse, la description du projet quitte le monde RTL et repose sur l'emploi de primitives propres à la famille de composants ciblée.
- Placement/routage :
  - Quartus utilise le résultat de la synthèse, place les primitives dans le circuit et assure les interconnections entre les blocs logiques.
  - C'est aussi à cet instant du design flow que les contraintes fournies par l'utilisateur sont prises en comptes :
    - orientation des signaux d'entrée et de sortie vers des broches choisies du composant (outil Pins Assigment),
    - choix de l'endroit où seront implémentés les différentes blocs fonctionnels du projet (outil Chip Planner).
- Analyse temporelle : elle permet de vérifier que le cahier des charges en matière de fréquence de fonctionnement peut être rempli par le design réalisé.
- Simulation temporelle : on y vérifie le fonctionnement de la puce finale en tenant compte des temps propagations des différents signaux à l'intérieur du composant. C'est l'étape ultime de contrôle.

1. Register Transfer Level : représentation niveau bascule et équation combinatoire d'un code VHDL synthétisable.

- Programmation du composant par téléchargement via le PC avec possibilité d'effectuer des tests directement en situation et de visualiser les signaux internes avec l'outil Signal Tap II Debugger.

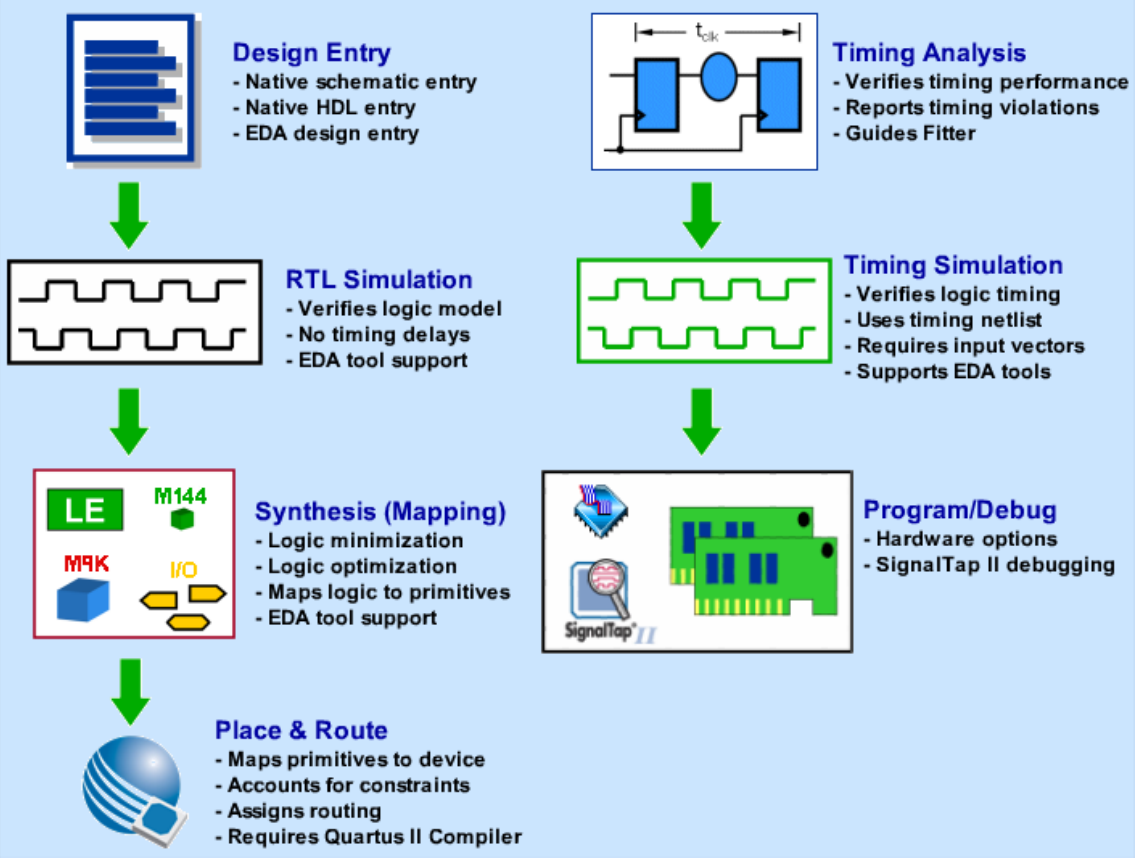

FIGURE 3.1 – Quartus II Design Flow (source : Tutorial Quartus)

2 Philosophie du développement : un exemple de design

2.1 Quelques consignes

Un petit dessin vaut mieux qu'un long discours. Lors de la conception d'un projet nécessitant la mise en oeuvre de différentes fonctions, on représentera toujours le projet par un schéma illustrant la structure fonctionnelle du projet. Ce schéma mettra en évidence les relations entre les différents blocs fonctionnels du projet, leurs interconnections, les connections avec l'extérieur ; chacuns des blocs fonctionnels pouvant être eux mêmes décrits par des schémas et, plus fréquemment dans notre cas, par du code VHDL, une machine d'état ... bref, tous les modes de saisie possibles.

Conséquence : certaines spécificités du VHDL (techniques de liaisons ou de générations de blocs par le VHDL) n'ont pas besoin d'être connues. L'outils de CAO en génère le code lui même. L'objectif de ce cours se limitera donc à l'apprentissage du VHDL pour la description de bloc fonctionnel s'appuyant sur l'utilisation d'un environnement tel que Quartus disposant d'une saisie de schéma hiérarchique. Cette approche, aussi minimaliste soit elle aux yeux d'un spécialiste, permet néanmoins d'aborder le VHDL en douceur. Les limites de ce cours sont claires : la réalisation d'un décodeur MPEG2 vidéo n'est pas un objectif !

Nous observerons les règles suivantes :

- La description en langage VHDL d'un bloc fonctionnel doit être la plus simple possible.
- Si une fonction paraissant trop complexe peut être décrite par un schéma hiérarchique utilisant des éléments plus faciles à implémenter, alors il en sera fait ainsi afin de facilité la conception des vecteurs

de test et de rendre ainsi le test plus exhaustif.

- L'utilisation de portes logiques résultant de mises en équations hasardeuses et au schéma peu lisible doit être proscrite : « Pensez et décrivez vos projets en terme fonctionnel ! »

2.2 Un premier projet : comptage et affichage des secondes de 0 à 9

2.2.1 Approche fonctionnelle

Cahier des charges : on souhaite réaliser un compteur de seconde élémentaire ne comportant qu'un seul chiffre est dont le résultat doit être affiché. On dispose pour cela d'un signal d'horloge précis de 50 MHz et d'un afficheur 7 segments.

Notes :

- Vu la simplicité du cahier des charges, il ne serait pas choquant de décrire l'ensemble du projet en un seul fichier VHDL où les différents aspects fonctionnels seraient de toute façon abordés séparément en différents endroits du code. Il y a donc également une raison plus pédagogique de proposer ce découpage fonctionnel sous la forme d'un schéma hiérarchique.
- On utilise généralement les FPGA pour réaliser des tâches rapides. Un comptage à l'écehelle d'une seconde se réalise davantage avec un microcontrôleur.

Méthode de résolution : il serait tout à fait possible d'écrire un programme VHDL unique prenant en entrée le signal d'horloge et fournissant directement 7 informations représentatives de l'état des segments de l'afficheur pour retranscrire visuellement le nombre de secondes. Néanmoins, le programme obtenu serait très spécifique et ne laisserait pas apparaître les concepts de développement de façon visible. Une seconde approche consiste à voir le problème par ses spécifications fonctionnelles afin d'en déduire une forme de solution résultant de l'association de blocs fonctionnels assez classiques et à l'implémentation simple.

Des pistes de résolutions :

- L'objectif étant de compter des évènements chaque seconde, il faut donc constituer un signal d'horloge de 1 Hz. L'horloge disponible sur la carte de développement DE2 est de 50 MHz et il apparaît donc judicieux de réaliser un premier bloc fonctionnel de division de fréquence avec un rapport de 50 000 0000 caractérisable comme suit :

```
Fonction : diviseur de fréquence (c'est une forme de compteur).
Entrées :
  clk_50MHz : le signal d'horloge à 50 MHz de la carte DE2.
  reset     : un signal d'initialisation.
Sortie :
  clk_1s : le signal de 1 Hz qui sera utilisé pour attaquer le
           compteur de secondes.
Codage : VHDL.
```

- A partir de ce signal de 1 Hz, il reste donc à compter de 0 à 9 et à afficher le chiffre compté sur un afficheur 7 segments. Deux éléments fonctionnels se détachent naturellement de la description du travail restant.

```
Fonction : compteur BCD (comptage cyclique de 0 à 9).
Entrées :
  clk_1s : signal d'horloge d'entrée du compteur BCD.
  reset  : signal de remise à 0 asynchrone du compteur pour les
           phases d'initialisation.
Sortie :
  bcd[3..0] : vecteur 4 bits (pour autoriser un codage d'entiers de 0 à 9).
Codage : VHDL (on pourrait utiliser le générateur de MegaFonctions
         intégré à la saisie de schéma de Quartus).
```

```
Fonction : décodeur BCD vers 7 segments
           (fournit 7 signaux a, b, c, d, e, f, g pour attaquer
            les 7 segments de l'afficheur et faire apparaître
            le chiffre codé par BCD sur 4 bits).
Entrée :
  bcd[3..0] : vecteur 4 bits représentant l'entier à afficher.
Sortie :
  a, b, c, d, e, f, g : les 7 signaux correspondants aux 7 segments
                        de l'afficheur.
Codage : VHDL (même si les bibliothèques Altera basées sur
         MAX PLUS 2, l'ancêtre de Quartus, intègrent ce genre de symbôles).
```

Remarque : il existe de nombreux outils (Quartus en intègre) permettant de générer des éléments relativement standards (pour des fonctions simples telles que des additionneurs, compteurs ...) avec une liberté de paramètrage autorisant une juste adaptation aux besoins de chacun. Au delà des fonctions simples, de nombreuses sociétés proposent des modules de propriétés intellectuelles (IP pour Intellectual Property) pour des fonctions beaucoup plus sophistiquées telles que des coeurs de microprocesseur (les IBM Power PC et Strong ARM sont fréquents), des décodeurs audio ou video, des interfaces Ethernet ...

2.2.2 Réalisation sous Quartus II

La réalisation sous Quartus amène donc à l'écriture de 3 sources VHDL et d'un schéma hiérarchique. Ce dernier est présenté sur la figure .

Remarque : la réalisation complète de ce projet fera l'objet d'une séance de travaux pratiques.

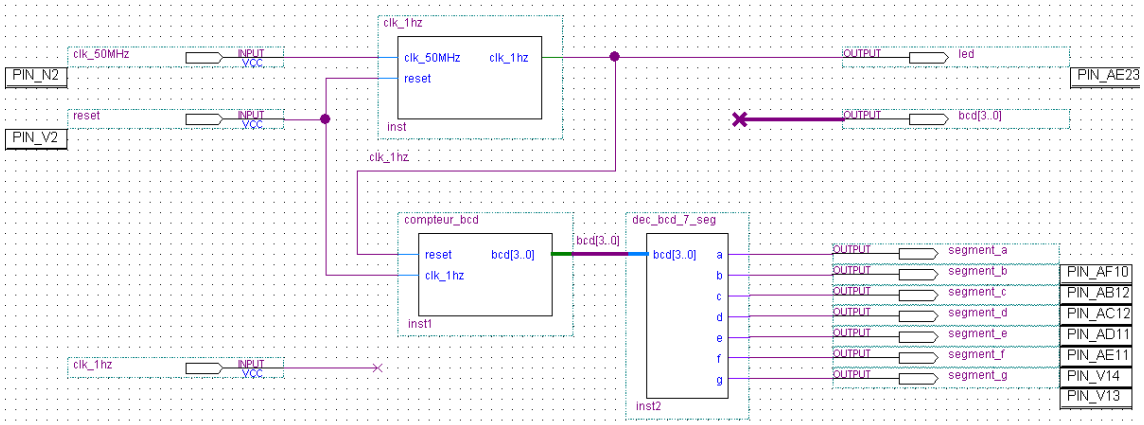

FIGURE 3.2 – Schéma hiérarchique du compteur de secondes

3 Utilisation de Quartus II sur un exemple simple

3.1 Introduction

Afin de se concentrer sur les éléments essentiels de la création d'un projet sous Quartus II, le sujet de démonstration est une simple porte ET à deux entrées. Pour que l'approche hiérarchique à partir d'un schéma fonctionnel soit abordé, cet exemple n'utilise pas une porte existante à l'intérieur de l'éditeur de schéma mais un symbôle représentant un ET à deux entrées et dont le comportement va être décrit dans un programme écrit en VHDL.

3.2 Création du projet

Quartus II démarré, on crée un nouveau projet : *File -> New Project...*

La première boite de dialogue permet de fixer le nom du projet et son lieu de stockage (voir figure 3.3).

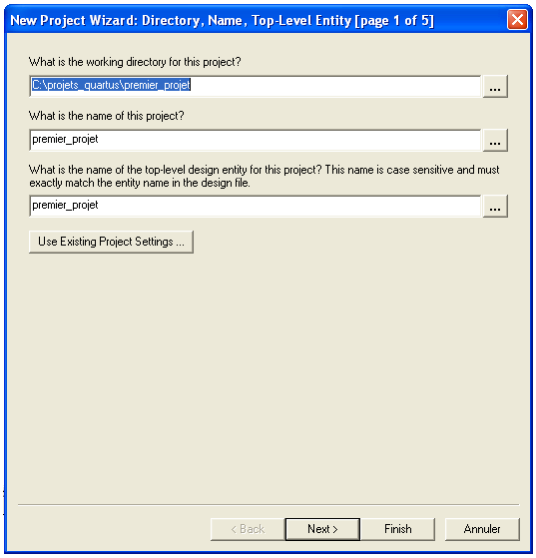

FIGURE 3.3 – Création de projet sous Quartus II

Dans la seconde, il est possible d'ajouter des fichiers déjà existants au projet. Ne rien ajouter et cliquer sur le bouton *Suivant*.

La troisième fenêtre de dialogue permet de choisir le FPGA ou CPLD ciblé. Le composant utilisé dans l'ensemble de ce cours est le CYCLONE II EP2C35F672C6 (famille : CYCLONE II ; nombre d'éléments logiques : 35k ; boitier : F pour FGA ; nombre de broches : 672 ; grade de vitesse : 6).

Une fois choisi, on en termine avec l'assistant de création de projet en cliquant sur *Finish*.

L'environnement de Quartus II (figure 3.4) : il se découpe en 4 parties. 3 sont visibles au début du projet. Une fenêtre Project Navigator qui contient toutes les informations relatives au projet dont les détails sur la hiérarchie du projet, les fichiers le constituant, les différentes unités définies (entités, architecture, schémas, machines d'états ...). La fenêtre Tasks présente les différentes tâches du Design Flow avec la possibilités d'accéder à l'ensemble des processus de traitement de Quartus, les comptes rendus associés. Pour finir, la fenêtre Messages avec ses onglets de filtrage multiples permet d'être informé en continu des informations, averstissements et erreurs apparaissant lors de l'exécution des diverses tâches.

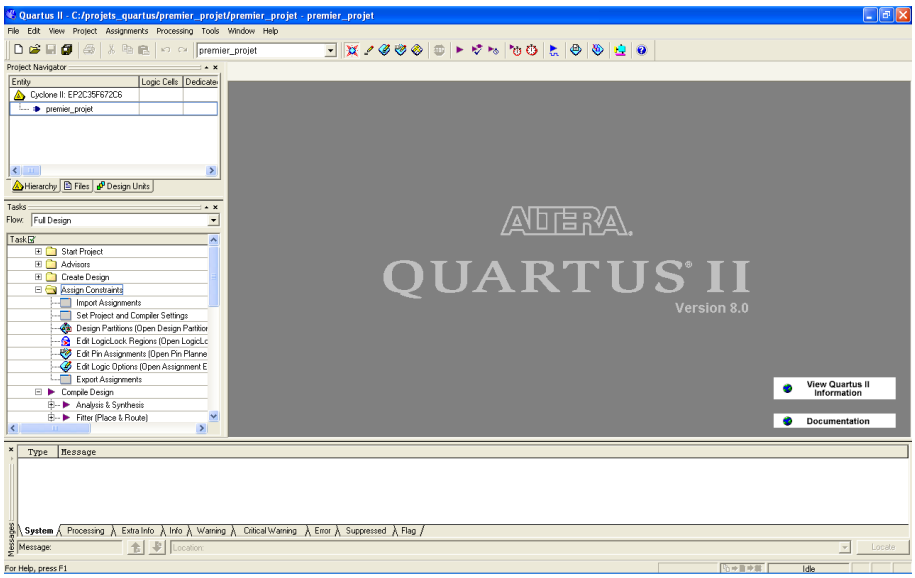

FIGURE 3.4 – Environnement de développement de Quartus II

Avant d'aborder la suite du projet, quelques remarques et rappels s'imposent :

- Dès lors que le projet possède plusieurs blocs fonctionnels, son design repose sur une hiérarchie avec un schéma à l'apparence la plus fonctionnelle possible en son sommet (top hierarchy). Cela signifie qu'on évitera de placer des structures nébuleuses à base de portes logiques élémentaires dans ce schéma du sommet hiérarchique.
- Lors du design sur Quartus II d'un projet hiérarchique, le concepteur commence par décrire les blocs fonctionnels de plus bas niveau en premier (en les testant séparément). Le schéma donnant la structure fonctionnelle situé au sommet de la hiérarchie est saisie en dernier. Il faut avoir à l'esprit que ce schéma ne peut de toutes façons être dessiné qu'à partir du moment où le concepteur dispose des symbôles adéquats, dont l'existence est subordonnée à la réalisation préalable des différents éléments fonctionnels du projet.

3.3 Premier source VHDL

Le cycle de développement obéïssant à une philosophie montante, on commence donc par la conception des différents blocs VHDL (puisque le choix est fait de décrire ces blocs en VHDL dans le cadre de ce cours). Dans le cadre de ce premier exemple, le choix est fait de se limiter à un projet ne contenant qu'un seul source VHDL décrivrant le comportement d'une porte ET.

Pour ajouter un nouveau source VHDL : *File -> New...* et on choisit un source de type VHDL (il est intéressant de remarquer l'étendue des possibilités en matière de source).

Les détails de l'écriture du code à proprement parler seront vus dans la suite du cours.

Taper le code suivant et le sauver (*File -> Save As...* en lui donnant le nom *porte_et.vhd*).

```
-------------------------------------------------------------------------
library ieee;                   -- Chargement de la bibliothèque IEEE
                                -- qui est indispensable en VHDL
use ieee.std_logic_1164.all;    -- Utilisation du paquetage
                                -- std_logic_1164 en totalité
use ieee.numeric_std.all;       -- Utilisation totale du paquetage
                                -- numeric_std
-------------------------------------------------------------------------


-------------------------------------------------------------------------
-- Description externe de la porte_et2
-- On indique la liste des entrées et la liste des sorties
-- Le mot clé entity déclare le début de la description externe
-- dont le nom est porte_et2
-------------------------------------------------------------------------
entity porte_et2 is port (
  -- Input ports
  entree_a : in  std_logic;     -- au nom de l'entrée succède : puis
                                -- le sens in pour entrée et son type
                                -- std_logic (un type binaire amélioré)
  entree_b : in  std_logic;
  -- Output ports
  sortie   : out std_logic      -- bien entendu, on utilise out pour
                                -- les sorties
);
end porte_et2;


-------------------------------------------------------------------------
-- L'architecture apporte une description interne comportementale
-- du bloc fonctionnel déclaré dans l'entité.
-------------------------------------------------------------------------
architecture comportement_porte_et2 of porte_et2 is
begin
  -- Dans cette exemple, on saisie simplement une équation logique.
  sortie <= entree_a and entree_b;
end comportement_porte_et2;
```

Remarque : l'étudiant confronté à l'apprentissage de nombreux langages peut rapidement se trouver submerger par la mémorisation précise requise par les différentes syntaxes auquelles il se heurte. L'éditeur de Quartus propose donc un outil permettant d'accéder à des modèles pour les diverses instructions du langages VHDL (idem pour Verilog) ; modèles qu'il est possible d'inclure d'un simple clic dans le code source en cours d'édition. Cet outil (Template) est accessible via le menu *Edit -> Insert Template...* ou via l'outil  dans la palette d'outils de l'éditeur.

3.4 Vérification de la syntaxe du code

Outre un repérage des mots clés par des couleurs différentes, l'éditeur de code VHDL Quartus propose un outil de contrôle d'erreur très utile pour limiter la présence de coquilles et d'erreur d'inattention syntaxique dans le code.

Il est donc conseillé de l'utiliser (*Menu Processing -> Analyse current file* ou l'outil  dans la palette d'outils de l'éditeur).

3.5 Création d'un stimulus de simulation

Quand le code VHDL est correct en matière de syntaxe, il est ensuite nécessaire de le tester en lui appliquant différents vecteurs de test. Les systèmes de développement proposent en général deux solutions

pour réaliser ce test :

- L'éditeur graphique de vecteurs de test (Vector Waveform Editor) comme le propose Quartus II. C'est simple d'utilisation, visuel mais ne permet pas les tests les plus exhaustifs qui soient.
- Le banc de tests VHDL (réciproquement Verilog pour les concepteurs qui décrivent leurs blocs fonctionnels avec ce langage). Moins facile d'accès au premier abord, il offre une souplesse incomparable par rapport à l'éditeur graphique et permet des tests beaucoup plus complets. L'outil de simulation intégré à Quartus ne gère pas ces Test benchs mais permet cependant d'exporter un test graphique sous la forme d'un test bench VHDL qu'il suffit ensuite de retravailler. Pour finir, il suffit de le fournir à un outil de simulation externe (EDA Simulator). Dans notre cas, ModelSim Altera sera le simulateur tièrce partie mis en oeuvre pour les simulations à partir de vecteurs de tests fournis avec un fichier VHDL.

Pour créer un fichier de test graphique, atteindre le *menu File -> New...* et choisir *Vector Waveform File*. Dans la foulée, utiliser la commande *Save As...* pour enregistrer le fichier de test sous le nom *test_porte_et2*.

Afin de rendre les signaux d'entrées et de sorties accessibles à l'édition dans l'éditeur graphique, il faut réaliser une compilation de Design : *menu Processing -> Start Compilation*.

Ajouter des signaux d'entrée :

- Pour créer des stimuli sur les entrées, il faut ajouter les signaux d'entrées dans la colonne *Name* par un double clic dans cette même colonne (ou la commande *Insert -> Insert Node or Bus...* du *menu Edit*). Poursuivre par un clic sur le bouton *Node Finder...* de la boite de dialogue qui apparaît. Dans le second dialogue, sélectionner *Pins :all* dans la boite de liste *Filter* et lancer l'opération de filtrage par un clic sur le bouton *List*.
- Ajouter les signaux *entree_a* et *entree_b* dans la liste des signaux sélectionnés (*Selected Nodes*) en utilisant le bouton > . Puis valider par OK deux fois.

Donner un stimulus à un signal :

- Sélectionner le signal *entree_a* dans la colonne *Name*.
- Création d'un signal de type horloge pour le stimulus de *entree_a* : *Menu Edit -> Value -> Clock...*

  (ou cliquer sur l'outil de création d'horloge ). On remplit le dialogue d'horloge pour créer une horloge de 100 ns de période (*Period : 100 ns*).

Remarque : pour changer la durée de la simulation, utliser la commande du *menu Edit -> End Time...* .

Reprendre la même manipulation pour la seconde entrée avec une période de 200 ns.

Terminer en sauvant le fichier de test graphique.

3.6 Simulation du projet

Il est maintenant temps de simuler le projet. Pour cela, il convient préalablement de définir les paramètres de simulation en accédant à la boite de dialogue correspondante (*menu Assigments -> Settings* puis *Category Simulator Settings* ou également par la fenêtre de tâches, section *Verify Design -> Simulate Design -> Quartus II simulator -> Edit settings*).

- Choisir une simulation fonctionnelle (*Functional Simulation*) pour le mode de simulation (*Simulation mode*).
- Ajouter le fichier *test_porte_et2.vwf* pour les paramètres d'entrées de la simulation (*Simulation input*).
- Valider par OK.

Dans la fenêtre des tâches, lancer la simulation par un double-clic sur *Quartus II Simulator (Functional)*.

On obtient le résultat de la figure 3.5.

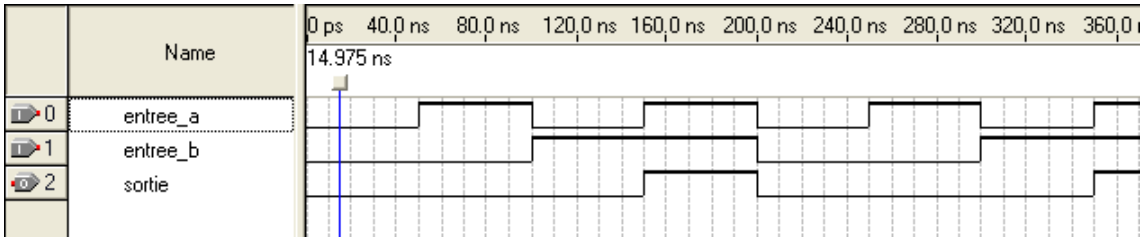

FIGURE 3.5 – Simulation fonctionnelle de la *porte_ et2*

Pour finir, reprendre les étapes précédentes avec une simulation temporelle (en choisissant Timing pour le mode de simulation). Les résultats obtenus doivent être identiques à ceux de la figure 3.6.

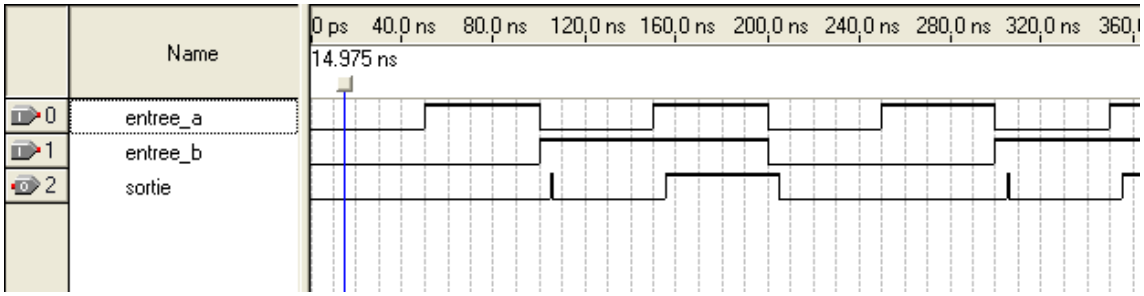

FIGURE 3.6 – Simulation temporelle de la *porte_ et2*

Remarque : on peut constater que le signal de sortie est différent de ce qui avait été obtenu dans le cadre de la simulation fonctionnelle. Deux phénomènes se manifestent : d'une part, un retard du signal de sortie (bien visible lorsque les deux entrées sont à '1') et d'autre part un comportement parasite repérable par la présence d'impulsion parasite (glitch) au moment où les signaux d'entrées se "croisent".