Antoine d'Hermies

ÉLECTRONIQUE NUMÉRIQUE

Architecture, VHDL, technologie des circuits programmables

DUNOD

Direction artistique : Élisabeth Hébert
Conception graphique de la couverture : Pierre-André Gualino
Mise en page : Lumina Datamatics, Inc.
Illustration de couverture : raigvi / shutterstock.com

11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-079435-5

Table des matières

1 Introduction et méthodologie

1.1 Introduction

Cet ouvrage s'adresse aux étudiants en BTS ou DUT, aux élèves ingénieurs ainsi qu'aux ingénieurs et techniciens qui cherchent un ouvrage de synthèse regroupant les connaissances indispensables pour mener à bien le développement d'un circuit numérique en utilisant le langage de description VHDL.

L'approche est résolument orientée vers la mise en œuvre pratique. C'est pourquoi en plus des bases théoriques, le lecteur trouvera de nombreux exemples concrets en VHDL qui illustrent tous les thèmes théoriques étudiés. Le développement en logique câblée est aussi présenté, car il permet de formaliser les aspects théoriques.

On trouvera de nombreux exemples de codes VHDL dans le livre. Chacun d'eux a été simulé et implémenté sur une carte de développement FPGA. Dans ces exemples, on s'est attaché à utiliser une large variété d'instructions pour montrer les diverses facettes du langage. Grace à un index qui recense les diverses syntaxes et instructions utilisées dans les exemples, le lecteur trouvera facilement un modèle utile pour ses propres développements.

Dans le chapitre dédié au langage VHDL, on s'intéresse essentiellement au VHDL synthétisable et aux instructions spécifiques à la simulation. Le lecteur trouvera toutes les informations nécessaires sur la syntaxe, la grammaire, les instructions ainsi que tous les détails utiles à l'étude d'un circuit.

Les aspects théoriques fondamentaux de la conception numérique sont développés par thèmes.

L'algèbre binaire introduit la numération et les différents codages usuels des informations utilisés en électronique numérique.

La logique combinatoire présente les opérateurs booléens et leurs propriétés. Les fonctions combinatoires sont étudiées en logique câblée ainsi qu'en VHDL.

La logique séquentielle est centrée sur la logique synchrone, car c'est la technique utilisée en conception numérique. Là encore, les différentes fonctions sont développées en logique câblée et en VHDL. Les composants historiques aujourd'hui obsolètes pour la plupart sont également présentés à titre d'exemples. Enfin, un paragraphe consacré à la logique asynchrone présente sommairement cette méthodologie spécifique.

Les machines d'état qui décrivent la partie contrôle d'un circuit sont largement détaillées. Les méthodes de descriptions, les architectures des machines de Moore, de Mealy

et les différents encodages des états sont agrémentés d'exemples en logique discrète et en VHDL.

Un chapitre présentant la technologie des circuits programmables permet au lecteur de découvrir les différentes familles de circuits et leurs technologies. On insiste plus particulièrement sur la famille des FPGA, qui est la plus importante. Les ressources internes de ces composants sont détaillées, donnant au lecteur des connaissances solides sur l'architecture de ces circuits.

Un chapitre présente les principes et les outils des suites logicielles proposées par les vendeurs de circuits programmables car l'informatique est indissociable de la conception de circuit.

Pour terminer le livre, une étude de cas détaille le développement complet d'un circuit en partant du cahier des charges jusqu'à l'implémentation dans un FPGA.

1.2 Circuit numérique : quel choix d'architecture ?

Dans le domaine des circuits intégrés numériques polyvalents, on trouve essentiellement 3 familles. Les processeurs qui sont des circuits à architecture fixe, les circuits à architecture configurable et les SOPC (*System On Programmable Chip*) qui intègrent un ou plusieurs processeurs et un FPGA dans un même boitier.

1.2.1 Les processeurs

Ces circuits intègrent principalement des unités de calculs, des unités de gestion des instructions, des données et de la mémoire, ainsi que des interfaces avec l'extérieur et de la mémoire interne. Pour exécuter leurs tâches, ils ont besoin d'un programme. Celui-ci est écrit dans un langage informatique, le langage C par exemple. Une fois traduit en langage machine, le programme est rangé dans une mémoire reliée au processeur.

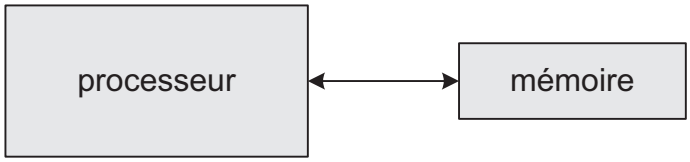

Figure 1.1 – Architecture à processeur

Les instructions sont exécutées séquentiellement les unes après les autres.

Par exemple, si on veut exécuter l'opération $S = A + B + C + D$ avec un processeur n'ayant qu'une seule unité de traitement, on procèdera comme suit :

- Les données A, B, C, D et les résultats sont rangés dans des registres internes du processeur.
- On décompose l'instruction en une suite d'opérations élémentaires :
  - $R1 = A + B$.
  - $R2 = C + D$.
  - $S = R1 + R2$.

► Le déroulement suit la Figure 1.2.

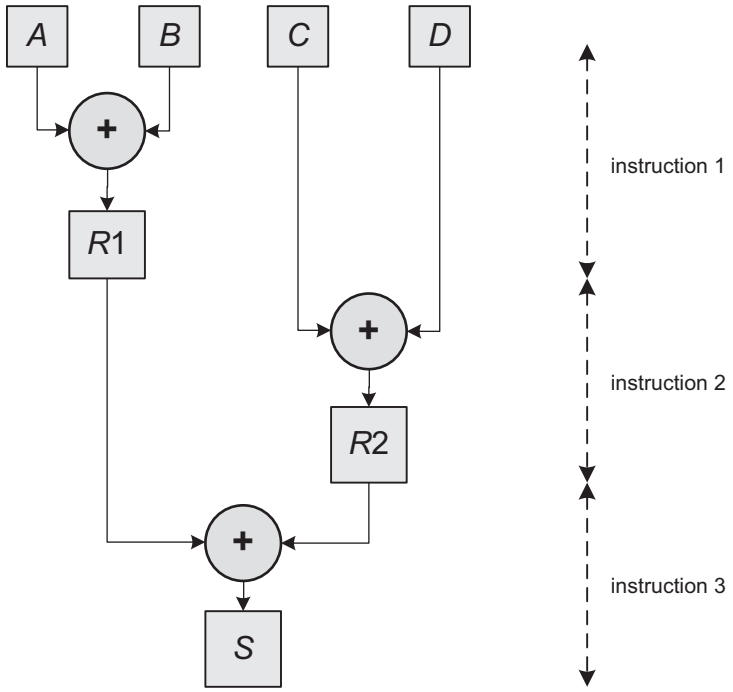

Figure 1.2 – Déroulement d'un algorithme

Il n'est pas possible d'obtenir directement le résultat, car on ne dispose dans le processeur que d'une seule unité d'addition. C'est pourquoi on a exécuté 3 instructions à la suite. Comme les instructions sont séquencées par une horloge, le seul moyen d'obtenir le résultat plus rapidement est d'augmenter la fréquence de l'horloge. Mais on n'obtiendra pas le résultat avant 3 cycles d'exécution d'une instruction. Notons également que le temps d'exécution d'une instruction représente plusieurs périodes d'horloge système. Si notre processeur avait eu 2 unités de calcul, les 2 premières additions auraient pu se dérouler en parallèle.

La mise en œuvre efficace des processeurs nécessite de connaître l'architecture spécifique de ces circuits. De plus, il est indispensable de maîtriser un langage informatique de programmation.

1.2.2 Les circuits programmables

Ils permettent d'assembler des portes logiques, des registres et autres fonctions préexistantes dans le circuit, pour réaliser diverses architectures.

Par exemple, reprenons le calcul précédent :

$$S = A + B + C + D$$

En implémentant 3 additionneurs, on réalise ce calcul en 2 couches, et l'enchaînement des opérations ne nécessite pas de stockage intermédiaire des résultats partiels.

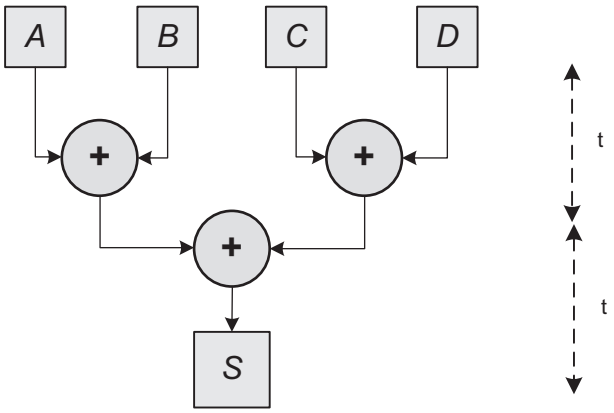

Figure 1.3 – Architecture parallèle

Le temps d'exécution se limite au temps de traversée des 2 couches d'additionneurs, qui est inférieur au temps d'exécution d'une instruction d'un processeur. Ces composants sont les seuls qui permettent de réaliser du parallélisme vrai.

Les circuits programmables forment une famille variée dont le circuit emblématique est le FPGA. Leur mise en œuvre nécessite une bonne connaissance de l'architecture numérique bas niveau. On décrit les architectures numériques avec des langages de description de matériel comme le VHDL ou le Verilog.

Dans cet ouvrage, nous nous intéressons uniquement à ces circuits et à leur mise en œuvre.

1.2.3 Les SOPC

Cette famille est en pleine expansion. Les fabricants intègrent de l'architecture configurable et des processeurs dans le même boîtier. Cela permet de bénéficier des avantages de chacune des familles précédentes. L'architecture de ces circuits peut se schématiser comme représenté en Figure 1.4.

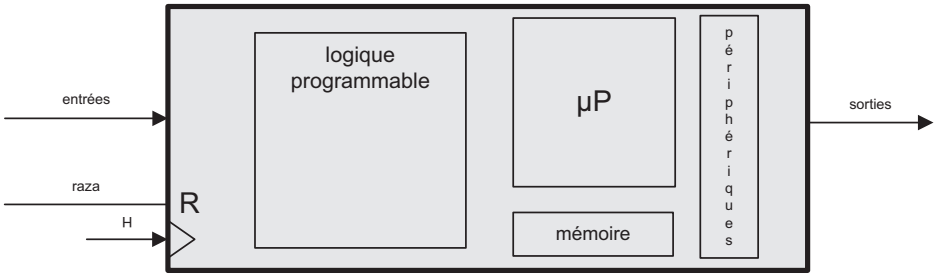

Figure 1.4 – SOPC

Comme les SOPC mélangent l'architecture configurable et les processeurs dans un même circuit, leur utilisation efficace nécessite d'avoir une double compétence informatique et architecture numérique.

1.3 Les différents niveaux d'étude d'un circuit numérique

Dans l'étude d'un circuit numérique, il existe plusieurs niveaux de descriptions possibles qui vont du plus bas niveau au plus haut niveau.

1.3.1 *Transistor level*

Le niveau transistor (*transistor level* en anglais) est le plus bas niveau. C'est à ce niveau que l'on étudie la construction microélectronique des composants.

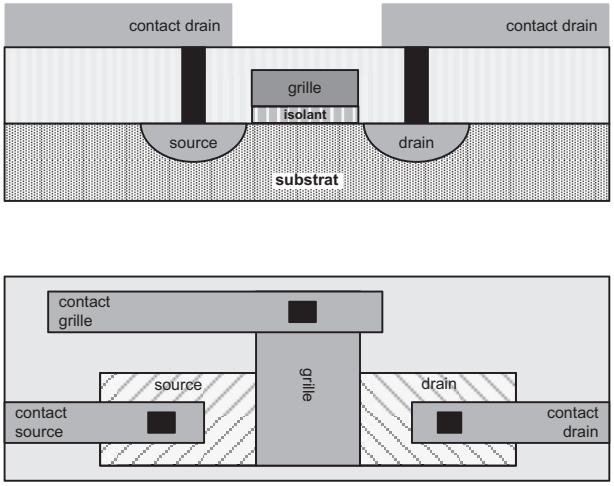

Figure 1.5 – Transistor MOS

- On a représenté les principales couches d'un transistor MOS vu en coupe et vu de dessus. Sur la coupe, on remarque la grille isolée du substrat et que le contact de grille n'a pas été représenté.
- La finesse de la technologie se caractérise par la distance entre drain et source.

À ce niveau d'étude, on s'intéresse à la construction des portes logiques avec des transistors construits sur mesures et optimisés. Les cellules créées sont entièrement caractérisées tant sur le plan électrique que sur le plan temporel. Cela permettra de choisir la cellule la mieux adaptée à chaque utilisation. On appelle ces cellules précaractérisées des primitives. On les regroupe en bibliothèques qui seront utilisables au niveau supérieur.

1.3.2 *Gate level*

On trouve ensuite le niveau porte (*gate level* en anglais). On s'intéresse ici à la réalisation de circuit avec des portes logiques et non plus aux transistors.

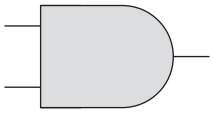

Figure 1.6 – Porte logique

On utilise des bibliothèques de portes logiques préconçues. Il suffit d'assembler des cellules de bibliothèques pour réaliser les fonctions logiques voulues. À ce stade, on est utilisateur de primitives et non plus concepteur de figures de bibliothèques.

C'est la technique utilisée pour réaliser les logigrammes des circuits numériques en assemblant des portes logiques. On appelle cela de la logique câblée, de la logique à composants discrets ou, plus simplement, de la logique discrète.

À ce stade, on traite chaque information indépendamment les unes des autres.

1.3.3 *Register Transfert Level*

Le niveau suivant est le niveau RTL (*Register Transfer Level*). On traite l'information de façon globale et non plus de façon unitaire. Le circuit est vu comme une succession d'alternances de registres de mémorisation et de couches de combinatoire.

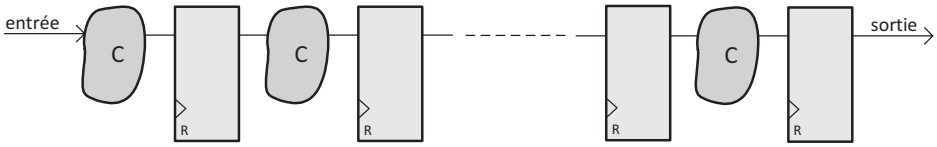

Figure 1.7 – Description RTL

Avec cette structure, on peut étudier tout ce qui se passe à chaque registre. Le circuit est vérifiable et observable en tous points.

Les circuits sont décrits avec un langage de description de matériel, qui est pour nous le VHDL. On construit une description du circuit, qui se traduira par de la logique grâce à l'outil de synthèse.

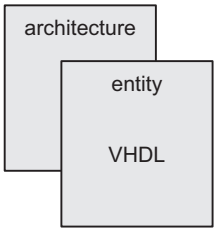

Figure 1.8 – Description VHDL

C'est à ce niveau que nous travaillons quand nous voulons décrire un circuit dans le but de le réaliser.

1.3.4 *Behavioral level*

Ce dernier niveau, appelé niveau comportemental (*behavioral level* en anglais) décrit le fonctionnement du circuit sans chercher à le réaliser.

$$y = e^x$$

$$y = \sin x$$

Figure 1.9 – Description comportementale

Cette étape permet de valider le concept du circuit et de modéliser son fonctionnement. Pour étudier le circuit, on peut utiliser des outils de modélisation de haut niveau (par exemple Matlab, un des plus connus) ou d'autres langages comme le System C, le System Verilog ou encore du VHDL non synthétisable, c'est-à-dire du VHDL qui ne peut être traduit par de la logique.

On notera qu'en VHDL, le niveau comportemental peut être confondu avec le niveau RTL si l'on utilise uniquement des fonctions synthétisables dans la description.

1.4 Constitution d'un système numérique

1.4.1 Les différentes fonctions d'un circuit numérique

Un système numérique est un ensemble de circuits électroniques qui traitent des données binaires entrantes et génèrent des sorties, elles aussi binaires, dont la valeur dépend des entrées et du traitement effectué en interne.

Figure 1.10 – Système numérique

On distingue 3 types de circuits, décrits ci-après.

Les circuits combinatoires

Quand les sorties ne dépendent que de la valeur instantanée des entrées, on dit que le système est combinatoire. Chaque combinaison des entrées agit directement sur les sorties.

Par exemple, prenons le circuit suivant qui possède 2 entrées et 4 sorties. À chaque valeur de la combinaison des entrées, une seule sortie est activée. Les autres sont inactives.

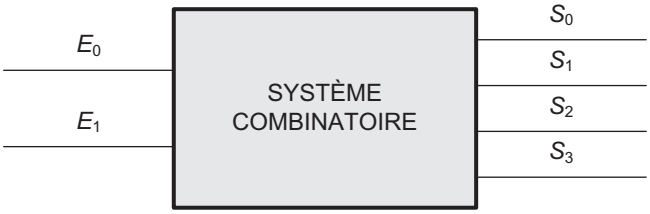

Figure 1.11 – Système combinatoire

On constate que dès que l'on modifie une valeur en entrée, les sorties changent.

Ce type de circuits sera détaillé dans le chapitre 4, consacré aux fonctions combinatoires.

Les circuits séquentiels

Si les sorties dépendent des valeurs des entrées mais aussi des valeurs des sorties, on dit que le système est séquentiel. Il y a une notion de mémoire de l'état du système dans ce type d'architecture. Les sorties dépendent des entrées présentes, mais aussi de l'historique du système.

Pour fixer les idées sur le système séquentiel, prenons l'exemple d'un compteur. Il évolue au rythme d'une horloge. La valeur qu'il va afficher au prochain top d'horloge dépend de sa valeur présente, mais aussi des entrées qui définissent son mode de fonctionnement.

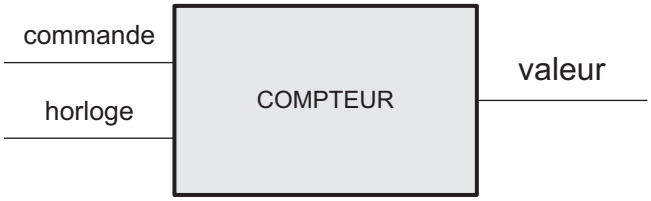

Figure 1.12 – Système séquentiel

- Sur le schéma bloc, le circuit présente 2 entrées et une sortie.
- L'entrée "commande" permet de choisir le mode de fonctionnement du circuit. Dans notre exemple, soit le compteur compte librement ("commande=1") soit son fonctionnement est suspendu et sa valeur ne change pas ("commande=0").
- L'entrée d'horloge règle la vitesse d'évolution du circuit.

La valeur future du compteur dépend de la valeur présente du compteur et de la valeur de "commande".

Supposons que nous soyons à la valeur actuelle 5. Si l'entrée "commande" est positionnée à '1', alors au top d'horloge suivant, la valeur de la sortie sera 6. En revanche, si "commande" est à '0', la valeur de la sortie ne change pas. Elle vaut toujours 5.

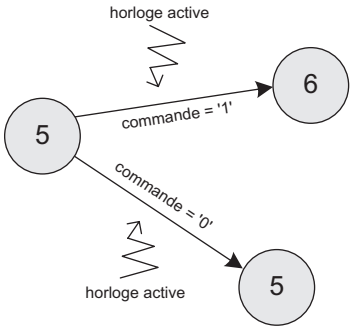

Figure 1.13 – Évolution du compteur

Ce concept sera précisé dans l'étude détaillée des systèmes séquentiels.

Les automates ou machines d'état

Un système numérique permet de réaliser des automates qui contrôlent l'évolution des tâches et le comportement du circuit. Mais les dénominations machines d'état ou

machines à états finis sont préférables. En effet, le terme automate peut prêter à confusion avec les automates programmables industriels (plus connus sous leur acronyme API). Ils ont une architecture interne très spécifique et sont dédiés à la réalisation d'automatismes et d'asservissements. Ils ont leurs propres méthodologies d'études et leurs propres outils de développement. Ces systèmes ne sont pas étudiés dans cet ouvrage.

Une machine d'état est une machine qui exécute une suite de tâches cycliques ou répétitives. Ces dispositifs sont utilisés pour contrôler des sorties du circuit, l'évolution du traitement des données entrantes ou gérer des circuits internes au système.

Prenons l'exemple d'un circuit de gestion des feux tricolores d'un carrefour routier.

Figure 1.14 – Gestion de feu tricolore

Sur la Figure 1.14, on remarque :

- L'entrée d'horloge qui permet de gérer le temps.
- L'entrée de mise en marche "on" du système.
- En sortie, les 3 leds : **vert**, **orange** et **rouge**. Les sorties actives correspondent aux feux allumés, celles qui sont inactives aux feux éteints.

Le cycle (vert – orange – rouge) se répète sans cesse avec des durées variables pour chaque couleur. C'est typiquement la tâche d'un automate.

Ces circuits sont détaillés dans le chapitre 7, consacré aux machines d'état.

1.4.2 Constitution d'un circuit numérique

Un système numérique est en général constitué d'un bloc qui traite les données et d'un bloc qui gère le déroulement des opérations.

On appelle partie opérative le bloc qui assure le traitement des données et partie contrôle le bloc qui gère l'évolution du circuit.

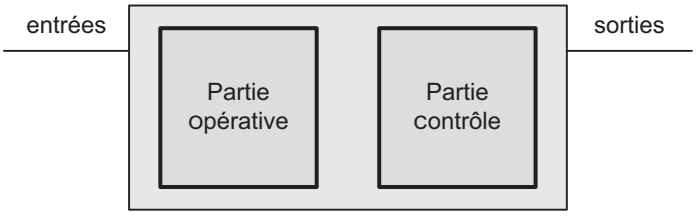

Figure 1.15 – Décomposition en partie opérative et partie contrôle

Précisons davantage cette architecture.

La partie opérative

Elle regroupe tous les calculs effectués par le circuit. Cela inclut des calculs internes effectués pour la partie contrôle et le traitement des données entrantes.

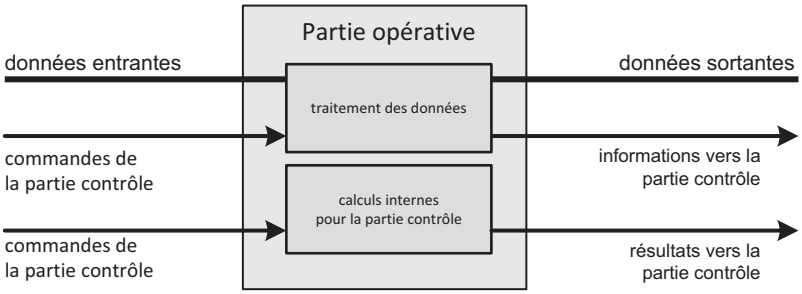

Figure 1.16 – Blocs de la partie opérative

- Le bloc de traitement reçoit les données entrantes et envoie les résultats en sortie du circuit. Il peut aussi envoyer des informations à la partie contrôle pour la renseigner sur l'évolution du traitement. De même, la partie contrôle peut également lui envoyer des commandes qui guident le traitement.
- Le bloc de calculs internes pour la partie contrôle communique également avec la partie contrôle. Il reçoit des instructions de celle-ci et lui envoie ses résultats.

La partie contrôle

Elle reçoit les informations des blocs de la partie opérative, celles des entrées de commandes ou des capteurs externes.

En sortie, elle envoie des commandes au bloc de traitement de données et au bloc de calcul interne de la partie opérative, ainsi que des informations en sortie du circuit avisant l'extérieur. On les appelle aussi des actions. Cela est représenté sur la Figure 1.17.

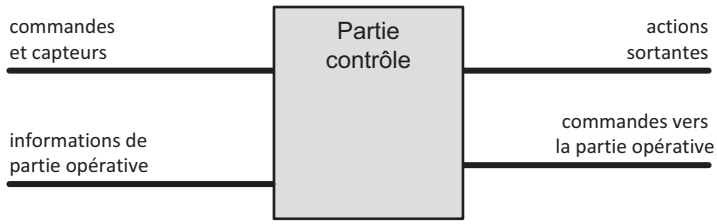

Figure 1.17 – Partie contrôle

Le circuit complet

On construit le circuit en regroupant les éléments de la partie opérative et de la partie contrôle.

- On remarque en bas à gauche de la Figure 1.18 le triangle qui matérialise l'horloge et le symbole R qui représente l'initialisation du circuit.
- "Sorties PC" représente les actions et les informations sur l'état du circuit à destination de l'extérieur.

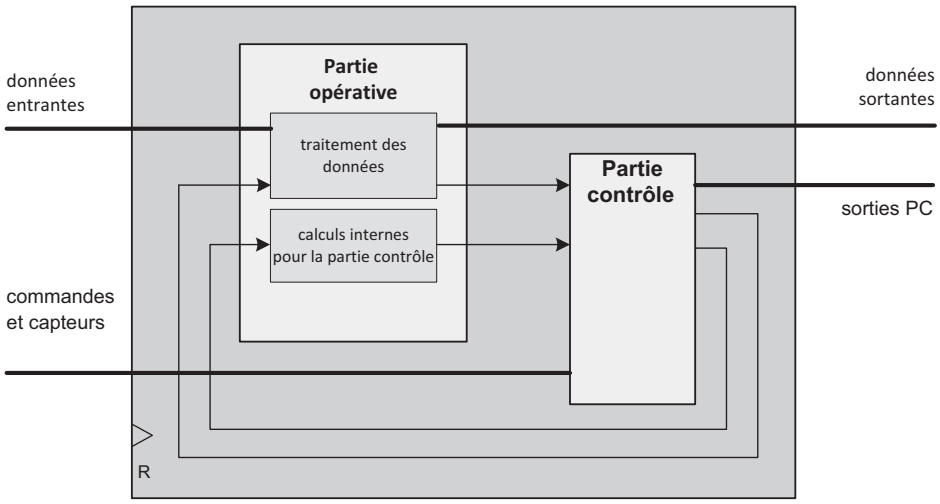

Figure 1.18 – Circuit complet

La Figure 1.18 représente une vision générale d'un circuit. Il y a de nombreuses variantes spécifiques à un projet donné.

Pour exemple, reprenons le cas du feu tricolore décrit dans la section 1.4.1 (Figure 1.14). Ce système a besoin de 3 temporisations pour calculer la durée d'allumage de chaque couleur. Elles sont commandées par la partie contrôle et elles l'avertissent quand elles ont terminé. Elles n'ont aucun lien avec l'extérieur du circuit. Les sorties du circuit (vert, orange, rouge) sont commandées par la partie contrôle. Comme il n'y a pas de données entrantes, le bloc de traitement correspondant n'existe pas. Ce qui donne le schéma du circuit décrit Figure 1.19.

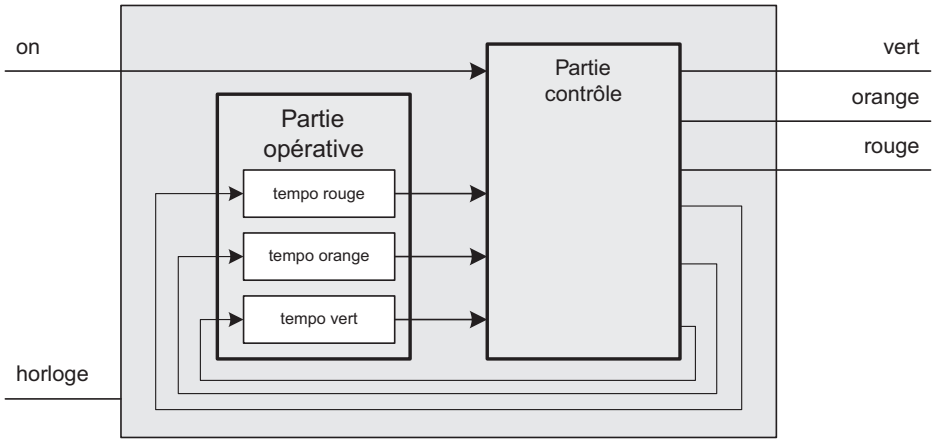

Figure 1.19 – Commande de feux tricolores

Pour réaliser le circuit, il faudra préciser les spécifications des temporisations pour pouvoir les décrire en VHDL.

On note qu'on peut réaliser ce circuit en utilisant une seule temporisation avec des seuils indiquant les durées de chaque tempo. Ce choix d'architecture est décidé par le concepteur.

1.5 Méthodologie d'étude d'un circuit numérique

1.5.1 Les 2 approches méthodologiques

Il existe 2 voies d'approche. La voie ascendante appelée *bottom-up*, et la voie descendante appelée *top-down*. La Figure 1.20 permet de comprendre à quoi cela correspond pour un circuit numérique.

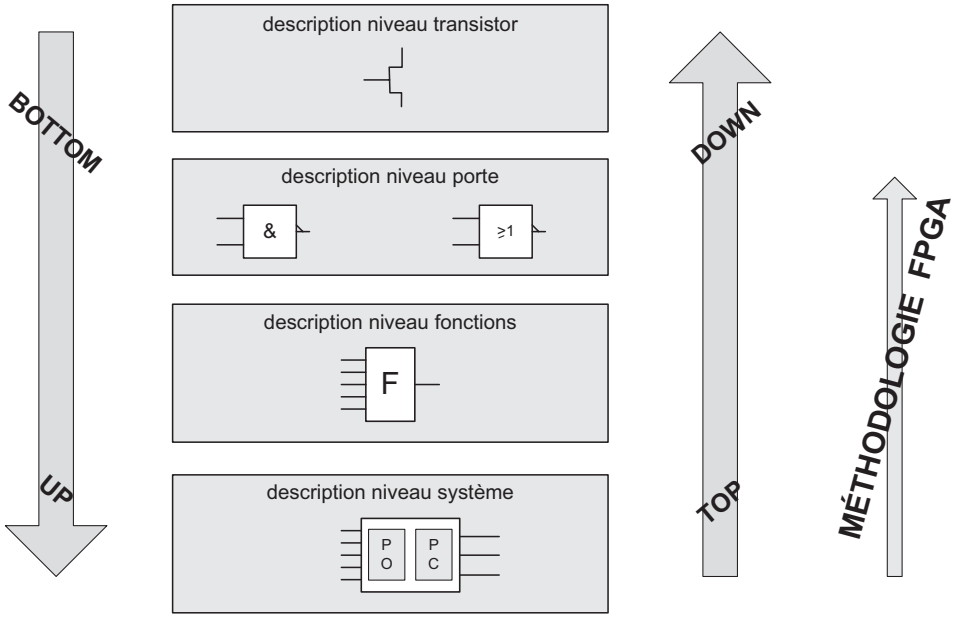

Figure 1.20 – Méthodologie *bottom-up* et *top-down*

L'approche *bottom-up*

C'est la méthode d'étude ascendante. On part d'un niveau inférieur pour remonter vers les niveaux supérieurs. Par exemple, on peut partir du niveau transistor pour fabriquer des portes, puis des fonctions. Cette méthode est utilisée par les fondeurs. Ils mettent au point leur technologie et développent leurs bibliothèques de primitives. Ils vendent ces bibliothèques aux utilisateurs qui veulent utiliser cette technologie pour fabriquer essentiellement des ASIC (*Architecture Specific Integrated Circuit*). Ce sont des circuits réalisés sur mesure pour des applications particulières, en assemblant des primitives parfaitement optimisées. Ces circuits sont les plus performants, mais ils ont un coût de développement et de fabrication très élevé et ils sont réservés à de grandes séries de fabrication.

On peut également commencer du niveau porte pour fabriquer des fonctions. C'est d'ailleurs cette approche que nous utilisons pour construire les briques de base des logiques combinatoire et séquentielle.

L'approche *top-down*

Avec cette méthode, on part du plus haut niveau de description. Une fois le concept validé, on construit une description RTL du circuit avec un langage de description de matériel. Ensuite, on utilise un outil de synthèse logique qui permettra d'arriver au niveau porte. C'est cette approche qui est utilisée pour étudier un projet.

Cette méthode est également utilisée pour réaliser des ASIC, mais avec un flot de conception beaucoup plus complexe que celui des circuits programmables. Mais cette utilisation, qui déborde des problématiques abordées ici, ne sera pas traitée dans cet ouvrage.

1.5.2 Méthodologie d'étude d'un système numérique

Le contexte

L'objectif d'une entreprise est de vendre un produit fini. Elle doit faire des bénéfices et amortir l'investissement fait pour le développement du produit. En général, plusieurs entreprises concurrentes travaillent dans le même domaine et développent simultanément des produits similaires. La première qui mettra son produit sur le marché aura une avance considérable sur les autres. Le seul paramètre sur lequel l'entreprise peut agir est le temps de développement du produit, car c'est le principal vecteur qui influe sur ce temps de mise sur le marché qu'on appelle aussi le TTM (*Time To Market*).

Deux voies peuvent permettre de réduire le TTM, soit renforcer l'équipe de développement, soit utiliser des éléments du projet déjà réalisés par ailleurs pour réduire le temps de développement.

La première possibilité, qui vise à renforcer une équipe, exige de disposer des compétences disponibles ou de recruter des collaborateurs. De plus, il faut que le projet s'y prête.

L'autre possibilité est d'acquérir des parties du circuit déjà développées et de les adapter. On appelle ces blocs des IP (*Intellectual Property*). Cette manière de développer est très courante. De très nombreuses entreprises développent des IP et les commercialisent. Les vendeurs de circuits programmables proposent d'ailleurs avec leurs suites logicielles un catalogue d'IP gratuites et payantes.

Le fait d'utiliser une IP commerciale permet de gagner beaucoup de temps. En effet, elles sont validées et documentées et bénéficient d'un support technique.

D'ailleurs, les entreprises gèrent leurs produits comme des IP. Elles tiennent à jour des bibliothèques de leurs développements avec des documentations rigoureuses et très complètes. Cette gestion méthodique est indispensable si l'on veut pouvoir facilement réutiliser des éléments d'un circuit.

Le choix de la cible est guidé par des aspects économiques mais aussi techniques. En effet, la solution du circuit programmable est séduisante mais ne se prête pas à toutes les utilisations. Si cette famille ne peut répondre à des critères de consommation, de performances ou de sécurité par exemple, il faudra se tourner vers un ASIC ou un ASSP

(*Application Specific Standard Product*) qui est un ASIC d'usage plus général. Mais le coût de production et de développement est très élevé et leur TTM est beaucoup plus long que pour un FPGA.

La méthodologie

On utilise la méthodologie *top-down* adaptée à la conception numérique en suivant la stratégie du cycle en V utilisée dans les entreprises.

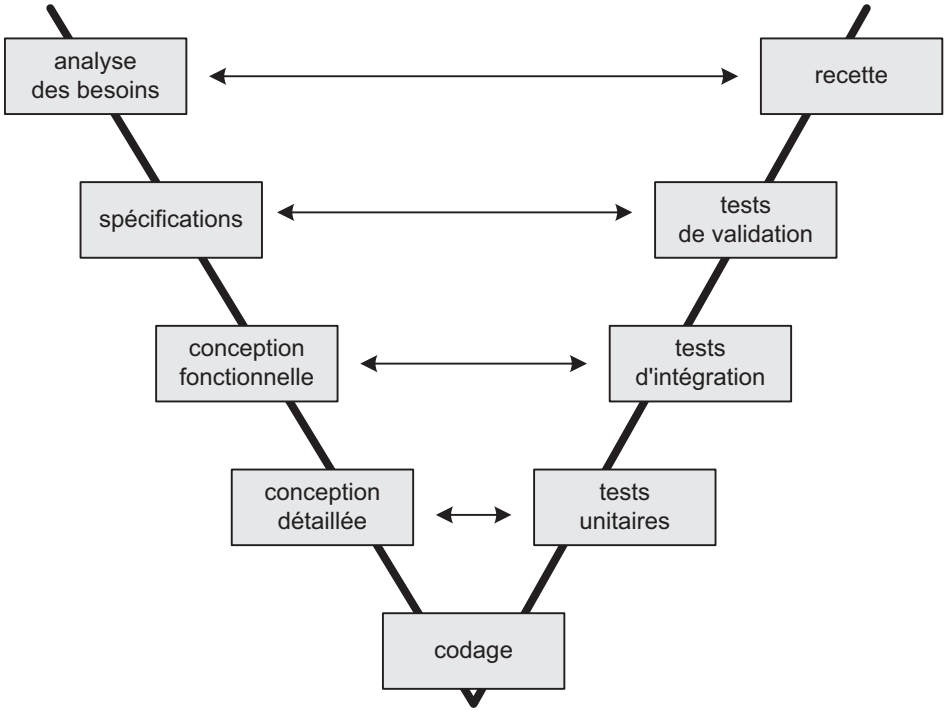

Figure 1.21 – Cycle en V

Une fois le projet construit sur le plan théorique et les spécifications établies, on rentre dans la phase de développement du produit.

La première chose à faire est de s'approprier le cahier des charges du projet.

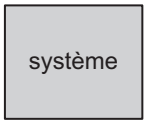

Figure 1.22 – Système

Pour cela, il faut recenser et définir clairement toutes les entrées et sorties du circuit. Cela permet de fixer le cadre de l'étude à mener et de bien comprendre le lien avec l'environnement extérieur. De plus, cette analyse permet souvent de lever des ambiguïtés du cahier des charges ou de préciser des éléments insuffisamment spécifiés.

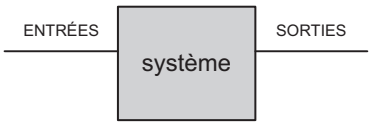

Figure 1.23 – Entrées sorties du système

Ensuite, on s'intéresse à l'architecture du circuit. On décompose le système en blocs fonctionnels. On trouve à ce premier stade l'organisation en partie contrôle et partie opérative.

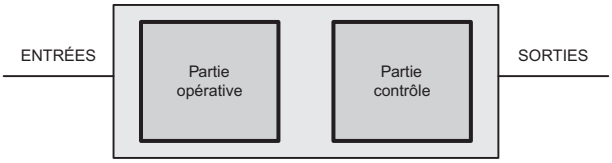

Figure 1.24 – Décomposition en PO/PC

On itère ce processus consistant à décomposer au maximum la partie opérative en blocs fonctionnels un peu à la manière des poupées russes. Certains blocs peuvent eux aussi être composés d'une partie opérative (PO) et d'une partie contrôle (PC).

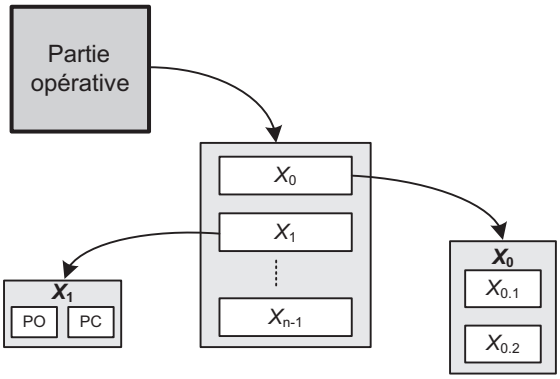

Figure 1.25 – Décomposition de la PO

Sur la Figure 1.25, on a représenté une partie opérative qui se décompose en n blocs. Parmi eux, le bloc X_1 se décompose en une partie opérative et une partie contrôle. Le bloc X_0 se décompose en 2 blocs $X_{0.1}$ et $X_{0.2}$.

Une fois qu'on a atteint un niveau de décomposition suffisamment abouti, on passe à la phase de développement des blocs. Ils sont décrits en VHDL et testés par simulation.

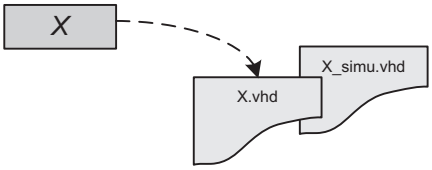

Figure 1.26 – Développement d'un bloc

Certains de ces blocs peuvent être des IP propres à l'entreprise ou acquises auprès d'entreprises tierces. Ces IP peuvent se présenter sous la forme de boîtes noires non modifiables (en VHDL crypté ou en netlist déjà synthétisée) ou en code source VHDL. Cela dépend du contrat passé avec le fournisseur d'IP. L'acquisition d'un code source offre le savoir-faire du vendeur, aussi est-il beaucoup plus cher.

Les blocs doivent ensuite être transformés en logique par l'outil de synthèse, puis validés suivant les règles de développement mises en place dans l'entreprise.

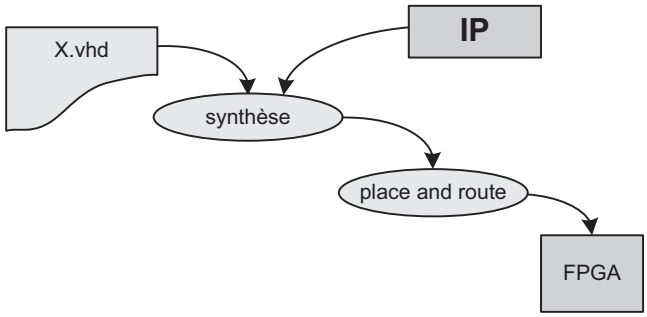

Figure 1.27 – Flot avec IP

L'utilisation détaillée des outils logiciels est présentée au chapitre 9.

On construit le système en assemblant les blocs validés et en vérifiant à chaque étape de la construction que tout est conforme.

2 Numération et codages binaires

2.1 Définition des variables

Nous étudions un ensemble où les variables sont binaires. Elles ne peuvent prendre que les 2 valeurs 0 et 1.

On appelle variable simple une variable qui a un seul élément binaire. On la nomme souvent bit (contraction de l'expression anglaise *binary digit*). Une variable multiple est une combinaison de bits. Ces variables peuvent être désignées par des noms d'usage dans certains cas. Les plus courants sont regroupés dans le Tableau 2.1.

Tableau 2.1

Français	Anglais	Définition
bit	*bit*	mot de 1 bit
quartet	*nibble, semi octet*	mot de 4 bits
octet	*byte*	mot de 8 bits
mot	*word*	mot de 16 bits
mot long	*long word*	mot de 32 bits
mot de *n* bits	*n bits word*	mot de *n* bits

2.2 Codes fondamentaux du numérique

2.2.1 Introduction

Il est indispensable de savoir manipuler les objets binaires et à quoi ils correspondent. Ils peuvent être représentés de différentes façons et il existe de très nombreux types de codages. Nous nous limiterons dans cet ouvrage aux codages usuels de l'électronique numérique.

2.2.2 Code binaire naturel

C'est le code directement issu de la définition mathématique de la numération en base 2. Une variable N décimale sur n bits s'exprime :

$$N = a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$$