Chapitre I

Outils de développement (IDE)

1. LE ROLE D'UN IDE

L'utilisation des circuits logiques programmables comme base des systèmes numériques est devenue courante. Cette utilisation nécessite un ensemble d'opérations plus ou moins complexes pour passer de la description d'une fonction à son implémentation sur la puce. Les étapes générales pour le développement sont :

- la modélisation du système ou de la fonction,
- la description généralement en langage évolué, graphique ou par équations de la fonction modélisée,
- la compilation, traduction complète ou partielle de la description en équations ou pseudo-équations,
- la simulation fonctionnelle pour valider la fonction,
- l'affectation des fonctions élémentaires aux ressources du circuit logique programmable cible et leurs interconnexions. Cette étape est appelée placement et routage (Place and Route) ou encore en un mot Fitting.
- la simulation temporelle pour valider les contraintes de temps du système global et de chaque cellule élémentaire (ressource),
- la programmation des équations dans le circuit logique programmable,
- le test sur site,
- l'exploitation.

Il est évident que tout ce processus ne peut pas être fait manuellement, ainsi tous les fabricants de circuits logiques programmables proposent une « suite » permettant de remplir toutes les fonctions précédentes et même plus avec une interface utilisateur conviviale. Ces outils sont appelés, en anglais IDE (Integrated Development Environment) et en français EID (Environnement Intégré de Développement).

Nous allons décrire les fonctions que l'on trouve sur la plupart des IDE. Il est clair que chaque IDE peut présenter des caractéristiques différentes. La documentation est généralement bien fournie avec une aide en ligne importante. La prise en main d'un IDE ne pose pas de problèmes spécifiques et le passage de l'un à l'autre prend peu de temps car souvent seule la présentation des menus est différente.

2. CARACTERISTIQUES GENERALES D'UN IDE

Le travail de base d'un IDE est de manipuler des fichiers, lancer des programmes de traitements de ces fichiers, effectuer automatiquement toutes les optimisations nécessaires ou demandées.

- **Notion de projet**

La plupart des IDE sont construits autour de la notion de projet. Un projet est un ensemble de fichiers manipulés par des outils pour créer d'autres fichiers. Par exemple les fichiers source sont compilés et associés à d'autres fichiers pour obtenir le fichier de simulation, le fichier de programmation à transférer dans le composant cible. Un outil est un programme de traitement donné qui permet, par exemple, de simplifier les équations de description d'un système. Cette notion de projet a permis au concepteur de se consacrer le plus possible aux tâches de modélisation et de description de la fonction souhaitée. Pour simplifier, on peut dire que le concepteur ne doit plus perdre son temps à penser d'abord à l'optimisation de l'implémentation de son système. Cette approche a une importance considérable dans la suite de cet ouvrage car nous allons mettre l'accent sur les modèles qui permettent de laisser les tâches liées à la mise en œuvre à des processus automatiques optimaux. L'enchaînement des outils peut être automatique ou spécifique.

- **Gestion de fichiers**

Les fichiers manipulés par l'IDE sont nombreux, on distingue :
- les *fichiers fournis* par l'utilisateur tels que les fichiers source, les fichiers de contraintes, etc.,
- les *fichiers générés* par les outils tels que les fichiers de netlist, les fichiers de simulation, les fichiers de rapports et les fichiers de programmation des composants.

- **Edition**

L'édition consiste à créer et à modifier des fichiers source (du texte, du symbole, du graphique).

- **Langages**

Tous les IDE offrent au moins les langages suivants : VHDL, Verilog, ABEL ou équivalent, et un langage graphique (qui permet la connexion des composants).

- **Passerelles entre langages**

Certains IDE permettent parfois de passer d'un langage à un autre au moyen de composants que l'on peut créer à partir d'une description d'un autre langage. Par exemple, avec Quartus II d'Altera, on peut générer un composant VHDL à partir d'une description Verilog ou d'un symbole graphique (composant) et vice versa.

- **Bibliothèques de composants**

L'IDE permet de créer des bibliothèques de composants. Mais il permet aussi d'exploiter les composants des fabricants tels que les LPM (Library of Parametrized Module) qui optimisent aussi bien les ressources propres à leurs composants que des fonctions spécifiques (bloc multiplicateur, bloc de division, bloc de calcul DSP ou bloc de calcul en flottant).

- **Choix du composant cible**

Pour exploiter au mieux le composant cible, il faut l'indiquer à l'IDE. Trois niveaux de choix de composant sont possibles. On peut laisser à l'IDE, soit la latitude de choisir le composant le plus adapté de manière automatique, soit imposer une famille de composants, soit encore on peut imposer directement le composant.

- **Contraintes**

Les contraintes sont des spécifications sur les performances attendues du système.
On peut distinguer deux classes de contraintes :

- les contraintes temporelles : elles agissent sur les différents temps que l'on rencontre dans les circuits logiques programmables,
- les contraintes d'espace : elles concernent l'assignation des broches et l'occupation de la surface par les cellules utilisées dans la réalisation des fonctions.

Le fonctionnement d'un circuit logique programmable dépend des temps de propagation dans les différentes couches logiques. Ces temps contribuent à limiter la fréquence maximale de fonctionnement.

- **Compilation**

La compilation est l'étape où les outils sont exploités pour analyser la syntaxe des descriptions, traduire la description en ensemble de fichiers que l'on assemblera. Nous allons reparler de la compilation un peu plus loin dans ce chapitre.

- **Synthèse**

La synthèse est la phase de la compilation pendant laquelle l'outil produit les équations sous forme de netlists. Une netlist est une liste d'interconnexion de cellules effectuant des opérations élémentaires.

Lorsque la synthèse ne prend pas en compte le temps, elle produit une description uniquement fonctionnelle ou comportementale. La prise en compte de la cible introduit la phase de placement-routage, appelée Fitting. Dans cette phase, d'une part les équations sont assignées aux ressources du composant cible (placement) et le routage permet d'établir les liens entre ces ressources. D'autre part, les temps de propagation vont permettre une description plus complète conduisant aux évaluations de performances.

- **Simulation**

Après la compilation, on peut analyser le système décrit. Pour cela, on dispose de deux types de simulation :

- la simulation fonctionnelle ou comportementale et
- la simulation temporelle (on dit aussi post routage).

La simulation fonctionnelle ne nécessite pas une compilation complète alors que la simulation temporelle nécessite la connaissance du composant cible pour permettre une évaluation des temps de propagation, de la fréquence maximale de fonctionnement et du calcul de l'énergie consommée.

- **Transfert sur le circuit logique programmable**

On distingue deux classes de fichiers à programmer selon que le composant cible est un CPLD ou un FPGA. Dans le cas d'un CPLD, on programme un fichier représentant les fusibles à positionner. Dans le cas d'un FPGA, s'il est de type SRAM, on peut programmer le fichier directement dans le FPGA par l'intermédiaire de l'interface JTAG (Joint Test Action Group). Mais si l'alimentation électrique disparaît alors il faut recharger le FPGA. Pour palier ce problème, les constructeurs ont mis en place une mémoire dite mémoire de configuration qui est de type EEPROM. Le contenu de cette mémoire est transféré automatiquement dans le FPGA à la mise sous tension. On peut aussi recharger le FPGA par le contenu de la mémoire de configuration quand on le souhaite car une entrée spécifique est réservée à cet effet.

3. CREATION ET GESTION DE PROJET

3.1 Les fichiers d'un projet

La création d'un projet consiste à définir les éléments de base nécessaires à l'IDE pour mettre en place une description. Il n'est pas indispensable dans un premier temps de définir toutes les caractéristiques (composant cible, brochage, contraintes temporelles, …). Des valeurs par défaut peuvent permettre de débuter par une approche fonctionnelle. Au fur et à mesure de l'avancement du travail, il devient nécessaire de renseigner ces différents points. Dans la pratique, les éléments de base sont :

- le dossier du projet dans lequel l'outil va générer les fichiers de gestion du projet, les fichiers temporaires et les fichiers de programmation du composant cible.
- le nom du projet, qui sert de nom générique aux fichiers qui ne diffèrent que par leurs extensions.
- la famille de composant à considérer.
- les outils externes à lancer au cours du développement.

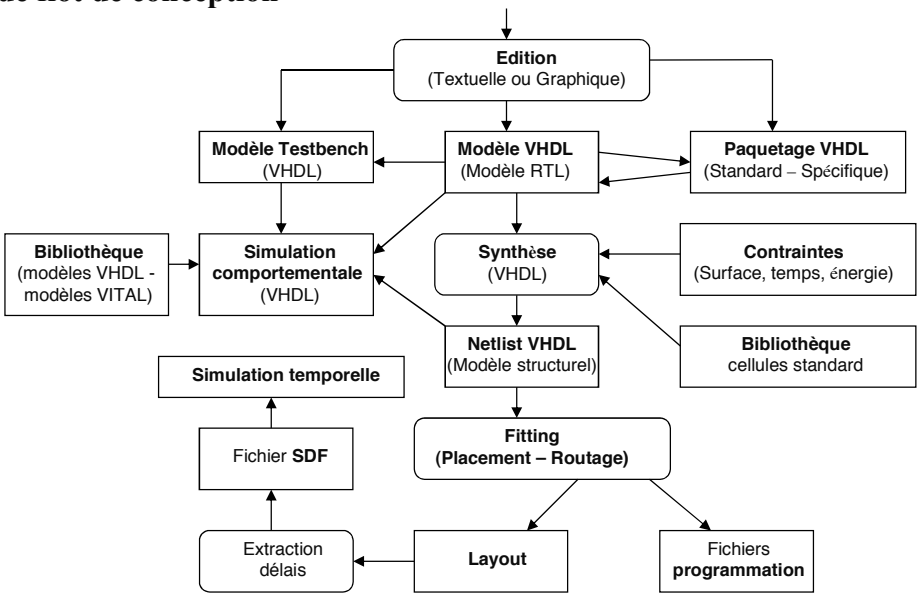
La gestion du projet va consister à :

- manipuler l'ensemble des fichiers du projet,
- enrichir progressivement le projet avec de nouvelles contraintes,
- lancer automatiquement les outils nécessaires à chaque phase du développement.

Nous donnons ci-dessous une liste non exhaustive des fichiers éditer et générés dans un projet:

Documents édités	*Documents générés*
Description de la cible (composant)	Fichiers résultats de l'analyse / synthèse
Description du brochage du composant	Fichiers résultats de la simulation fonctionnelle
Fichiers sources de description	Fichiers résultats du placement routage
Fichiers sources des stimuli de simulation	Fichiers résultats de la simulation temporelle
	Fichiers final de programmation

Exemple de flot de conception



Ce flot de contrôle pour lequel nous avons supposé l'utilisation du langage VHDL est général et peut s'appliquer à d'autres langages de description. Nous n'allons pas détailler toutes les explications pour ce flot parce que d'une part les opérations citées sont souvent enchaînées de manière automatique et parce que d'autre part, nous voulons rappeler la démarche générale.

Après l'édition de la description, les premières phases de la compilation permettent de générer un modèle de type RTL (Register Transfer Level). Ce dernier utilise la connexion de composants de base comme ET, OU, NON, Multiplexeur, registres, etc. pour définir un système numérique.

La phase de synthèse va générer une netlist qui est un modèle structurel du système. Les premières optimisations peuvent être réalisées par la synthèse. La netlist associée à des bibliothèques donne la possibilité de mener une simulation comportementale. Ensuite la connaissance des contraintes, comme le composant cible par exemple, permet de lancer la phase de Fitting qui est constituée de deux parties dépendantes, le placement et le routage, menées souvent de concert. Si l'opération arrive à terme, alors les fichiers de programmation du composant sont générés et l'extraction des délais permet la simulation temporelle qui prend en compte les caractéristiques réelles du composant. Un des résultats majeurs de cette phase est le temps de propagation du chemin le plus long qui limite la fréquence maximale de fonctionnement du système. Précisons que cette fréquence maximale peut être parfois améliorée par un réaménagement des blocs logiques du système.

Remarque :

Le fichier SDF (Standard Delay Format) contient les différents délais calculés pour les chemins rencontrés dans la description.

3.2 Les principales étapes de gestion d'un projet

3.2.1 La description

- **Description d'un système : Saisie de la description (Edition)**

Pour décrire un système, plusieurs approches sont possibles qu'il est parfois possible de combiner. On peut classer ces différentes descriptions en 3 catégories :

- **Description graphique : schéma logique / schéma bloc**

La description par schéma bloc est une description graphique. Chaque sous-fonction, considérée comme un composant, est décrite à l'aide d'un schéma logique. C'est une vue structurelle qui est hiérarchique par définition. On peut ainsi faire une description ascendante ou descendante. Puisque les autres langages de description peuvent être transformés en composants ou symboles, on peut donc les combiner. Des fonctions logiques prédéfinies (portes, bascules, compteurs, décodeur, mémoires, additionneurs, …) sont considérées comme disponibles. La description schématique permet aussi d'utiliser tout autre composant disponible dans une bibliothèque connue.

Remarque :

Certains outils offrent les moyens de décrire directement des machines à états et de réaliser leur synthèse.

- **Description textuelle : langages de description**

Plusieurs langages HDL sont inclus dans les IDE. Les plus courants sont : VHDL, Verilog, SystemVerilog et souvent un langage propriétaire comme ABEL de Lattice et AHDL d'Altera.

Une description textuelle peut être transformée en un symbole graphique que l'on peut utiliser lors de la description graphique. De même, on peut transformer un symbole graphique en un composant d'un des langages supportés par l'IDE.
Certains outils permettent d'éditer des fichiers de contenu de mémoire qui seront directement programmés dans la ressource mémoire d'un FPGA par exemple.

- **Bibliothèque de composants des fabricants (LPM) et IP**

La plupart des fabricants de circuits logiques programmables ou des éditeurs d'outils de développement (EDA = Electronic Development Automation) mettent à disposition des composants génériques complexes stockés dans une librairie appelée **LPM** (Library of Parameterized Modules). Ces composants vont des fonctions simples aux plus complexes. Ils sont optimisés pour les architectures de ces fabricants et ils exploitent les spécificités et les fonctions spéciales intégrées (Mémoires, PLL, Multiplieurs, etc.).
L'IDE permet de générer sous forme de composant configurable, l'élément LPM dans un des langages supportés. Par exemple, avec Quartus II d'Altera, on dispose du menu Megawizard pour cet effet.

De même, les composants appelés **IP** (Intellectual Property) sont développés et mis à disposition avec des autorisations payantes d'utilisation. Ils sont considérés comme des blocs fonctionnels matériels exploités comme tout composant.

3.2.2 La Compilation

La compilation permet d'analyser syntaxiquement les fichiers sources du projet en cours (fichiers de description, fichiers de contraintes, etc.) et de produire un ensemble de fichiers permettant la simulation ainsi que la programmation finale du circuit.

La compilation se décompose en 4 étapes principales (le résultat de chaque étape est contenu dans des fichiers distincts).

Analyse et Synthèse (Analysis & Synthesis) :
- Vérification des fichiers de description (syntaxe, cohérence, …).
- Simplifications logiques et synthèse des différentes descriptions en un seul fichier résultat. Vérification de la possibilité d'intégrer la description finale dans le composant choisi (vérification entre autre du nombre de portes nécessaires, bascules, éléments mémoires, nombre de broches, …).

Intégration – placement / routage (Fitter) :
- L'intégration affecte chaque fonction logique à un ensemble de ressources (utilisation optimale des ressources disponibles sur le composant choisi). L'objectif étant de respecter les contraintes fonctionnelles et temporelles du projet. L'intégration recherche à optimiser l'interconnexion entre les ressources.
- Il est possible de contraindre le système à utiliser certaines ressources.
- Lorsque les ressources disponibles sur le composant sélectionné sont insuffisantes, l'étape d'intégration génère un message d'erreur précisant ces ressources manquantes.

Assemblage :

- L'assemblage est la dernière étape de la compilation. Elle traduit le fichier résultat de l'intégration en un fichier de programmation du composant.
- Plusieurs formats de fichier de programmation sont envisageables en fonction de la cible choisie : programmation directe du composant, programmation de la mémoire de configuration du composant, …

Analyse temporelle :

L'analyse temporelle est basée sur le résultat de l'intégration. Cette étape vérifie et valide les performances temporelles (horloges, temps de propagation, …) de la logique synthétisée dans le composant choisi.

3.2.3 La synthèse

A partir des différentes descriptions, l'analyse et la synthèse :

- construisent une base de données,
- vérifient dans un premier temps la cohérence du projet (cohérence en termes de fonction logique). Cette base de données sera utilisée pour optimiser les équations logiques.
- optimisent la logique en utilisant plusieurs algorithmes pour réduire au minimum le nombre de portes et utiliser l'architecture du dispositif programmé aussi efficacement que possible.

3.2.4 La simulation

On distingue deux types de simulations : fonctionnelle et temporelle.

Simulation fonctionnelle

Cette simulation basée sur le résultat de l'analyse et de la synthèse des différentes descriptions, permet de vérifier le comportement logique du système décrit. Cette simulation peut s'effectuer sur l'ensemble du système synthétisé ou sur une partie.

Simulation temporelle

Cette simulation permet à la fois de vérifier le comportement logique du système (simulation fonctionnelle) mais aussi d'étudier les situations temporelles critiques. Elle n'est possible qu'après la phase de placement - routage. Elle décrit le comportement réel (en tenant compte des différentes contraintes temporelles ou d'espace,…) du système synthétisé.

Cette simulation prend davantage de temps que la simulation fonctionnelle. En contre partie, elle permet de valider complètement le fonctionnement du système décrit.

Les fichiers de simulation

Fichiers de stimuli :

Les IDE disposent d'outils de dessin permettant de tracer les chronogrammes des signaux d'entrée des systèmes à simuler. Après calcul, l'outil complète les chronogrammes avec les sorties calculées en fonction de la description.

Fichier Test bench : ce fichier contient

- une instance du système à simuler,
- la liste des entrées commandées avec des valeurs choisies,
- la liste des sorties attendues.

Ce fichier peut être décrit notamment en Verilog et en VHDL.

L'outil IDE utilise ce fichier et après calcul génère les résultats de la simulation sous forme de chronogrammes.

4. LES AVANTAGES D'UN IDE

L'IDE permet de traiter un système complet de manière virtuelle c'est-à-dire sans réalisation physique. Il donne la possibilité d'évaluer plusieurs solutions, de simplifier le circuit imprimé, d'anticiper les évolutions du système à moindre coût. Par exemple, un surdimensionnement va permettre d'ajouter des fonctionnalités nouvelles sans avoir à refaire le circuit mais tout simplement par une nouvelle programmation du composant. Nous donnons dans la suite quelques avantages que l'on peut trouver dans un IDE.

4.1 Choix du composant cible

Le choix sans a priori du composant cible peut s'effectuer entre : CPLD ou FPGA. Pour déterminer quelles ressources sont nécessaires, on peut procéder de la manière suivante :

- choisir une famille de composants et laisser l'outil guider le choix,
- puis affiner le choix pour définir le composant final.

Choix d'une famille de composant par défaut.

Le choix final du composant cible n'est pas nécessaire au début du projet. Il est possible de laisser l'outil choisir le composant le plus adapté au système décrit en termes de ressources ou de performances. L'étape d'analyse et de synthèse peut être effectuée sans que l'utilisateur définisse avec précision la famille - le composant - ou le brochage.
On a la souplesse de changer de famille pour comparer.

Critères de choix du composant final.

Plusieurs critères de choix doivent être pris en considération. Ci-dessous une liste de contraintes à satisfaire qui nous permettent de lister ces critères de choix :

→ Contraintes matérielles :
- nombres d'entrées/sorties disponibles
- type de boîtier
- consommation

→ Contraintes temporelles :
- temps de propagation
- fréquence maximale de fonctionnement

→ Ressources nécessaires :
- nombre de ressources / portes / bascules
- mémoire (RAM, ROM, FIFO, …)
- multiplieurs câblés
- fonctions spéciales : PLL …

4.2 Les contraintes

Les contraintes sont utilisées pour permettre à l'IDE d'opérer un certain nombre d'optimisations. Ces contraintes peuvent intervenir à différents niveaux de la conception.
On peut classer les contraintes en au moins 3 groupes :

- Les contraintes temporelles :
  Ce type de contraintes permet de définir la fréquence maximale attendue, les valeurs maximales et minimales des temps de pré-positionnement (tsu), de maintien (th), etc.