TECHNIQUES
DE L'INGÉNIEUR

Réf. : **E2455 V2**

Date de publication :
10 novembre 2018

Conception des systèmes VLSI

Cet article est issu de : **Électronique - Photonique | Électronique**

par **Frédéric ROUSSEAU, Olivier MULLER**

Mots-clés
systèmes multiprocesseurs | circuits intégrés | conception de circuits

Résumé Les progrès dans le domaine de la technologie des circuits intégrés associés au développement d'outils automatiques d'aide à la conception ont permis l'émergence de systèmes capables de contenir sur un seul circuit plusieurs dizaines de milliards de transistors (VLSI). Cet article présente l'ensemble des étapes de conception de ces circuits numériques (modèles, langages) et les étapes automatisées de synthèses comportementale, logique et physique. Il aborde aussi la conception de systèmes sur des architectures multiprocesseurs et les perspectives d'évolution de ces systèmes intégrés.

Keywords
multiprocessor systems | integrated circuits | circuit design

Abstract Computer aided design tools and microelectronic technology evolution in the last decades have facilitated the emergence of integrated systems composed of tens of billions of transistors (VLSI). This paper presents the different steps of digital system design: models, languages, and automated behavioral, logic and physical synthesis steps. The reader will find as well some information on the design of multiprocessor systems and the perspectives of these integrated systems.

Pour toute question :
Service Relation clientèle
Techniques de l'Ingénieur
Immeuble Pleyad 1
39, boulevard Ornano
93288 Saint-Denis Cedex

Par mail :
infos.clients@teching.com
Par téléphone :
00 33 (0)1 53 35 20 20

Conception des systèmes VLSI

par **Frédéric ROUSSEAU**
Professeur des universités
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), TIMA, Grenoble, France

et **Olivier MULLER**
Maître de conférences des universités
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), TIMA, Grenoble, France

Cet article est la version actualisée de l'article [E 2 455] intitulé « Conception des systèmes VLSI » rédigé par Frédéric ROUSSEAU et paru en 2005.

*Le marché des circuits intégrés est toujours en essor avec une croissance mondiale à deux chiffres en 2017 comme l'indique le comité WSTS (*Word Semiconductor Trade Statistics*). Ce marché est tiré par des champs applicatifs en forte croissante tels que le* big data*, généralement géré par des centres de données (*data centers*), mais aussi l'internet des objets (*Internet of Things *ou IoT), et l'informatique dans les nuages (*cloud computing*). Les objectifs ont aussi changé ces dernières années pour répondre aux enjeux sociétaux, notamment environnementaux, avec la réduction de la consommation énergétiques des circuits ou systèmes intégrés, mais toujours en recherchant la performance tout en optimisant le coût. On parle maintenant d'efficience, qui peut se traduire comme l'efficacité à moindre coût.*

Dans ce contexte, certains types de systèmes intégrés ou non sur une même puce subissent une forte demande. Les systèmes embarqués, enfouis et mobiles sont cachés et se comportent comme de véritables ordinateurs (invisibles) [H 8 000]. Ils contiennent généralement un ou plusieurs processeurs, de nombreux périphériques (coprocesseurs d'aide au traitement, gestion des entrées/sorties, gestion de la communication), des éléments mémoire, et des composants dédiés au traitement intensifs, tels que les processeurs graphiques (GPU pour « Graphics Processing Unit *») ou reconfigurables tels que les FPGA (pour «* Field Programmable Gate Array *»). De telles puissances de traitement nous rapprochent du domaine du calcul haute performance, avec des architectures intégrant de nombreux processeurs, 72 processeurs pour le Tile-Gx72, 256 processeurs pour le MMPA de Kalray et 3 584 cœurs pour le Tesla P100 de Nvidia. De tels systèmes sont réalisés avec quelques dizaines de milliards de transistors.*

Les outils automatiques d'aide à la conception de systèmes ont évolué et rendent possible l'intégration de ces milliards de transistors sur un seul circuit. Les outils et les méthodes de conception ont depuis les années 1980 fait le pas vers des niveaux d'abstraction plus élevés.

Ces nouvelles méthodes s'appuient sur des techniques et des outils largement utilisés en conception de circuits, et proposent aux concepteurs de systèmes de se focaliser sur des choix d'architecture et de technologie. Les étapes finales de conception sont alors effectuées par des outils automatiques.

Ce présent article fait le point sur les méthodes et les techniques de conception de systèmes et circuits numériques, depuis la spécification du système jusqu'à l'obtention des masques permettant de réaliser physiquement le circuit.

La première section présente les méthodes, les modèles et les langages utilisés dans les différentes étapes de conception de systèmes. Un flot classique de conception est ensuite présenté. Certaines étapes de la conception de systèmes intégrés, notamment la synthèse comportementale, puis la synthèse logique et physique, sont détaillées. La dernière section est consacrée à l'évolution des systèmes intégrés, d'abord en présentant les systèmes monopuces, puis en donnant quelques perspectives des réseaux sur puce.

Le lecteur trouvera en fin d'article un glossaire des termes utilisés.

1. Méthodes et modèles pour la conception

1.1 Évolution : de l'ASIC au système massivement multiprocesseur

Depuis la fin des années 1980 et la compilation de silicium, les outils d'aide à la conception de circuits intégrés ont évolué de façon significative. L'idée principale est d'augmenter le niveau d'abstraction lors de la conception et de laisser des outils automatiques traiter les étapes liées aux plus bas niveaux, souvent très fastidieuses et répétitives.

La conception de circuits intégrés de plusieurs dizaines de milliards de transistors ne peut s'envisager qu'avec des **outils logiciels d'aide à la conception** (outils de CAO), alors que les premiers circuits intégrés au début des années 1970 pouvaient être dessinés à la main. Les **circuits intégrés développés pour une application spécifique** sont appelés **ASIC** (*Application Specific Integrated Circuit*) (figure **1**). L'augmentation de leur complexité, atteignant le million de transistors dans les années 1990, n'a été possible qu'avec une évolution des outils de CAO vers des outils automatiques de plus en plus puissants et interactifs.

Avec la complexité des applications dans les années 2000, notamment celles liées au multimédia (téléphonie mobile, jeux, DVD…), ces systèmes appelés **systèmes VLSI** évoluent vers les **systèmes multiprocesseurs** [2]. Il s'agit alors de systèmes constitués d'un assemblage de plusieurs processeurs et de composants spécifiques (nommés IP pour *Intellectual Properties* ou ASIC). L'idée sous-jacente était de compenser la faible évolution de la fréquence d'horloge des systèmes VLSI par une parallélisation de l'architecture pour effectuer plusieurs traitements en parallèle, à l'aide de ressources spécialisées. Pour des raisons de coût et de fiabilité, tous ces composants sont placés sur la même puce. Ce sont les systèmes monopuces (*system on chip*) qui ont constitué les principaux composants dans la période 2000-2010.

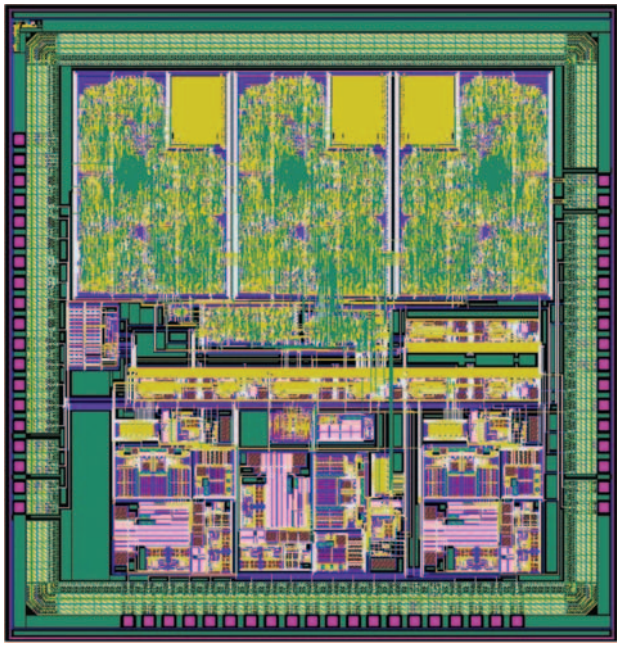


Figure 1 – Circuit intégré mixte numérique-analogique développé par le laboratoire IM2NP – UMR CNRS – Université d'Aix-Marseille et réalisé par CMP (CNRS) en 2011

Les besoins applicatifs sont encore et toujours en augmentation, et la tendance est depuis les années 2010 aux systèmes multiprocesseurs, ou massivement multiprocesseurs. Ces systèmes composés de plusieurs dizaines de processeurs, et de composants spécifiques, communiquent à travers des réseaux de communication performants, nommés réseaux sur puce (NoC pour *Network-on-Chip*) [3]. Ces réseaux ont généralement une structure architecturale hiérarchique, avec des grappes (*cluster*) de processeurs partageant un même espace mémoire (on parle alors de mémoire partagée), les grappes étant connectées par un réseau sur puce. On comprend facilement la différence entre ces architectures multiprocesseurs à travers les variétés de processeurs, leur nombre, leur connectivité, et la taille de la mémoire et des différentes mémoires cache. Cependant, c'est aussi sur des détails architecturaux que se démarquent ces systèmes : hiérarchie mémoire, notamment mémoire partagée globale ou partagée au sein d'une grappe, stratégie de cohérence de cache, performance des communications entre processeurs d'une même grappe, avec les processeurs d'autres grappes, performances du ou des réseaux de communication selon la topologie ou les algorithmes de routage ; un choix difficile mais qui dépend fortement de l'utilisation.

1.2 Concepts de base pour la modélisation des systèmes

Pour réaliser un circuit intégré, un concepteur raffine les spécifications de son circuit en descendant dans les niveaux d'abstraction. Le raffinement pour passer d'un niveau d'abstraction au suivant consiste à donner plus de détails de réalisation. On décompose ainsi le problème de conception en un ensemble de petits problèmes traités séparément, pour obtenir la réalisation physique du circuit.

Le diagramme en Y (figure **2**) [4] présenté par Gajski en 1983 [5] est toujours utilisé pour représenter les différentes vues et les différents niveaux d'abstraction d'un système VLSI. L'objectif est d'obtenir une représentation au niveau d'abstraction dit physique, qui permet la fabrication du circuit. Les trois axes représentent **trois domaines**. Le domaine **comportemental** correspond au comportement du circuit, sans information sur la manière de réaliser ce circuit. Le domaine **physique** nous renseigne sur les caractéristiques physiques de l'architecture finale qui composent le circuit. Enfin, le domaine **structurel** fait le lien entre les deux domaines précédents. Dans le domaine structurel, le système, ou le circuit, est décrit comme un ensemble de composants interconnectés réalisant le comportement spécifié dans le domaine comportemental.

La **conception d'un circuit** ou **d'un système** consiste à obtenir **une description physique** des masques du circuit au niveau physique en partant d'**une représentation comportementale de niveau système**.

À partir du diagramme en Y, on définit **trois tâches de conception**. On appelle **synthèse** (s) la transition entre le domaine comportemental et le domaine structurel. Le **raffinement** (r) correspond au passage d'un niveau d'abstraction vers un niveau plus bas en restant dans le même domaine. Enfin, l'**optimisation** (o) consiste à améliorer la qualité du circuit en minimisant son coût ou en augmentant ses performances, tout en restant dans le même domaine.

Le passage du domaine structurel au domaine comportemental est généralement appelé l'**analyse** et le passage du domaine structurel au domaine physique est la **synthèse physique** dont les principales étapes sont le placement, le routage et l'implantation. Enfin, le passage du domaine physique au domaine comportemental est l'**extraction**.

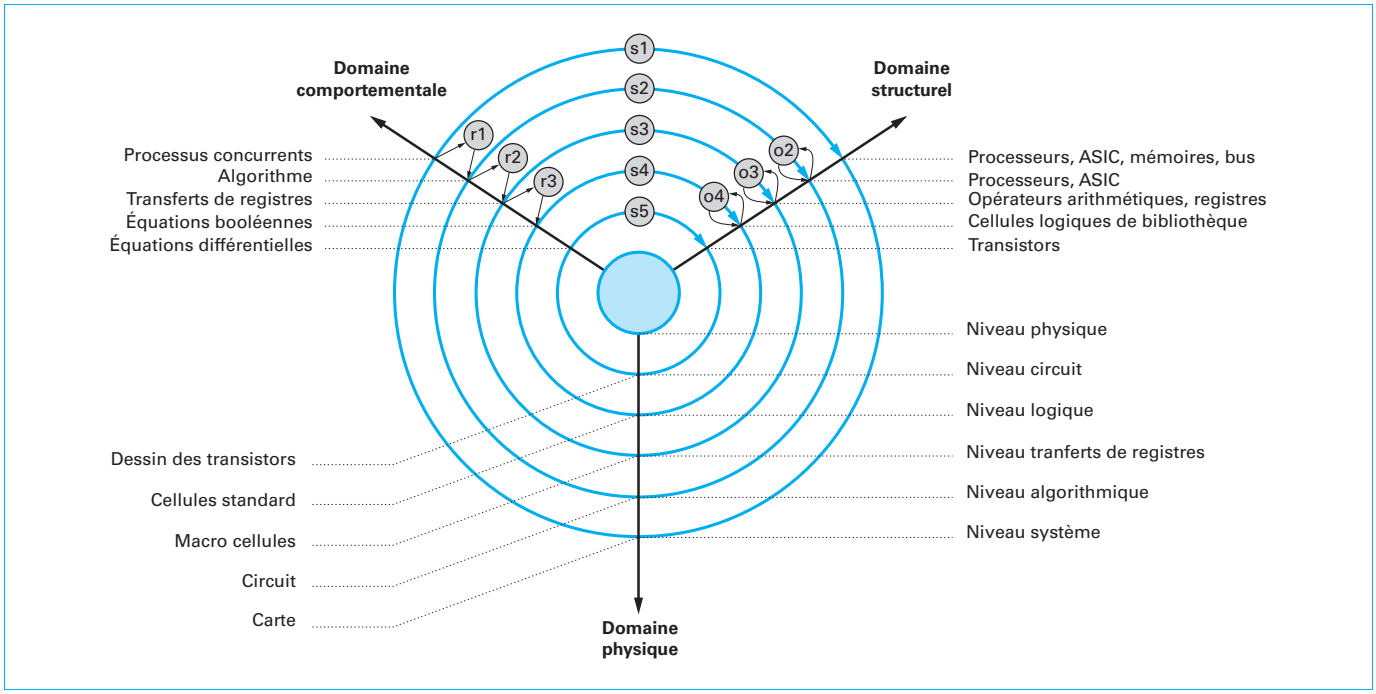


Figure 2 – Diagramme en Y

1.3 Niveaux d'abstraction

L'abstraction est un des **fondements de la modélisation et de la conception**. Elle permet de réduire la complexité du système en identifiant les caractéristiques intéressantes en vue d'une utilisation précise. Il s'agit de cacher les détails superflus sans nuire à la compréhension du problème, pour se focaliser sur d'autres difficultés. Selon l'état d'avancement dans la conception ou la modélisation, certaines caractéristiques ne sont pas essentielles à ce moment-là et peuvent donc être simplifiées.

On utilise généralement **six niveaux d'abstraction** (figure **2**). Au plus bas niveau, le niveau **physique** représente le dessin des masques du circuit. Il s'agit d'une liste de polygones représentant les différentes couches physiques du circuit (diffusion, polysilicium, métal…). Le niveau **circuit** est un schéma en transistors qui comporte aussi les caractéristiques physiques de chaque transistor (type, taille, paramètres). Le niveau **logique** est un ensemble de portes logiques simples telles que des fonctions NON, ET, NON ET, OU, NON OU, OU EX vérifiant des équations booléennes. Le niveau **transfert de registres** est un ensemble d'opérateurs arithmétiques pilotés par un contrôleur décrit en termes de registres et de circuits combinatoires. Des multiplexeurs et/ou des bus acheminent les données. Le niveau **algorithmique** est un ensemble de ressources matérielles complexes capable de réaliser les opérations décrites sous une forme algorithmique. Au niveau **système**, il s'agit d'un ensemble de processus concurrents. Il est utilisé pour vérifier les fonctionnalités et les relations temporelles entre les composants.

1.4 Différentes étapes de conception

■ À tous les niveaux d'abstraction, on peut définir les **étapes de synthèse**. La synthèse de niveau **système** (**s1** sur la figure **2**) consiste à transformer une spécification de niveau système (processus concurrents) en une architecture à base de composants complexes (processeurs, ASIC, mémoires, bus…). La synthèse **algorithmique** (**s2**) transforme un algorithme en une représentation structurelle utilisant des registres, des multiplexeurs et des unités arithmétiques et logiques pour réaliser les opérations. Les opérations sont séquencées par un contrôleur. La synthèse de niveau **transfert de registres** (**s3**) est principalement la synthèse de ce contrôleur, mais aussi des opérateurs arithmétiques et logiques qui sont dimensionnés et convertis en portes logiques. La synthèse **logique** (**s4**) permet de générer la vue structurelle du circuit. On s'appuie alors sur une bibliothèque de portes logiques existante (bibliothèque d'un fondeur ou cellules de FPGA) pour trouver la correspondance avec les équations booléennes (conversion technologique). La synthèse de niveau **circuit** (**s5**) consiste à transformer un ensemble d'équations en un schéma en transistors.

■ Dans le domaine comportemental, le **raffinement** qui permet de passer du niveau système au niveau algorithmique (**r1**) consiste à décomposer le problème de conception d'un système en un ensemble de spécifications des différents composants. Le passage du niveau algorithmique au niveau transfert de registres (**r2**) transforme la spécification algorithmique en une partie opérative pour le transfert et le calcul des données et une partie contrôle. Le passage du niveau transfert de registres au niveau équations booléennes (**r3**) revient à décomposer le contrôleur et les opérateurs en un ensemble de machines à états finis décrites au niveau logique.

■ Les **optimisations** les plus classiques sont l'optimisation du contrôleur (**o3**), en réduisant le nombre d'états et en codant les états pour réduire la logique associée. Les minimisations au niveau logique (**o4**) sont les simplifications des équations booléennes pour réduire la surface finale du circuit. Une optimisation avancée (**o2**) concerne la mise en œuvre d'architecture pipeline.

1.5 Précision temporelle

La notion de temps n'apparaît pas dans le diagramme en Y. Cette **notion de temps est essentielle**, car plus on descend dans les niveaux d'abstraction lors de la conception, plus cette caractéristique doit être précise et proche du comportement physique du circuit final. On définit différents niveaux d'abstraction pour le

temps. Au plus haut niveau, on utilise généralement l'**ordre sur des événements**. Ensuite, le **cycle d'instruction** représente le temps nécessaire pour réaliser une opération complexe. Le **cycle d'horloge** est un niveau d'abstraction inférieur représentant le temps de calcul et de transfert de données d'une instruction élémentaire. Au plus bas niveau, correspond le **temps physique** (équations différentielles décrivant les charges et décharges de condensateurs).

1.6 Difficultés de conception

La conception de systèmes consiste à passer **d'une description comportementale** de niveau système (c'est généralement la spécification du système à concevoir) **à une description physique** en suivant les étapes de raffinement, d'optimisation et de synthèse décrites précédemment. Le chemin à suivre dans la conception n'est pas unique et il dépend des connaissances du concepteur, des outils et des composants disponibles, du type de circuit visé, ainsi que de la méthode de conception, et il peut être différent selon les parties du système.

Une première difficulté de conception réside dans la **composition des domaines et des niveaux d'abstraction**. En effet, plusieurs domaines sont associés au même niveau d'abstraction, ce qui signifie qu'il existe plusieurs descriptions pour un même niveau. Il est aussi possible de représenter un système comme un ensemble de composants, chacun décrit à un niveau d'abstraction différent. À cela s'ajoutent différents niveaux d'abstraction possibles pour le temps. Il est donc difficile de définir un flot de conception unique et universel.

Une autre difficulté intervient dans les différentes synthèses. En effet, celles-ci s'avèrent triviales quand il s'agit de produire un modèle structurel, sans se préoccuper de performances. Mais la recherche de performances et d'optimisations conduit généralement le concepteur à introduire des concepts qui ne sont pas explicitement présents dans la description comportementale pour produire la description structurelle (pipeline, éléments de bibliothèque spécifiques…). La modification de la description initiale est alors nécessaire. Par exemple, au niveau algorithmique, l'utilisation de pointeurs donne souvent des résultats meilleurs que l'utilisation de variables. De même, la notion de pile pour mémoriser des informations n'apparaît pas dans les algorithmes.

De plus, les étapes dites de synthèse dans la conception de systèmes VLSI regroupent plusieurs étapes de raffinement, synthèse, analyse, et optimisation, selon les parties du système, ce qui rend difficile la compréhension des flots de conception.

Avec **les outils automatiques d'aide à la conception** actuellement disponibles, on peut définir un ensemble d'étapes pour la conception de systèmes numériques synchrones (type de circuits le plus courant) qui correspond au flot de conception le plus classique (figure **3**). Le point de départ est une description de niveau système dans le domaine comportemental. Une étape de **raffinement** appelé **partitionnement logiciel/matériel** (§ 2.3) permet d'obtenir une description algorithmique (**r1** sur la figure **2**) correspondant aux parties matérielles qui seront réalisées par un ASIC. Ensuite une étape complexe, appelée **synthèse comportementale**, correspond d'abord à une étape de raffinement (**r2**), puis à une étape de synthèse (**s3**) et à une étape d'optimisation (**o3**). La **synthèse logique** correspond à un raffinement (**r3**), à une synthèse (**s4**) et à une optimisation (**o4**) qui s'applique soit au résultat de la synthèse comportementale, soit à la description algorithmique. La **conception physique** est le passage vers le domaine physique pour obtenir les masques du circuit.

1.7 Langages utilisés pour la conception

De nombreux langages ont été développés pour couvrir le cycle de conception. On peut les classer en plusieurs catégories selon l'utilisation faite ou le type d'application visée.

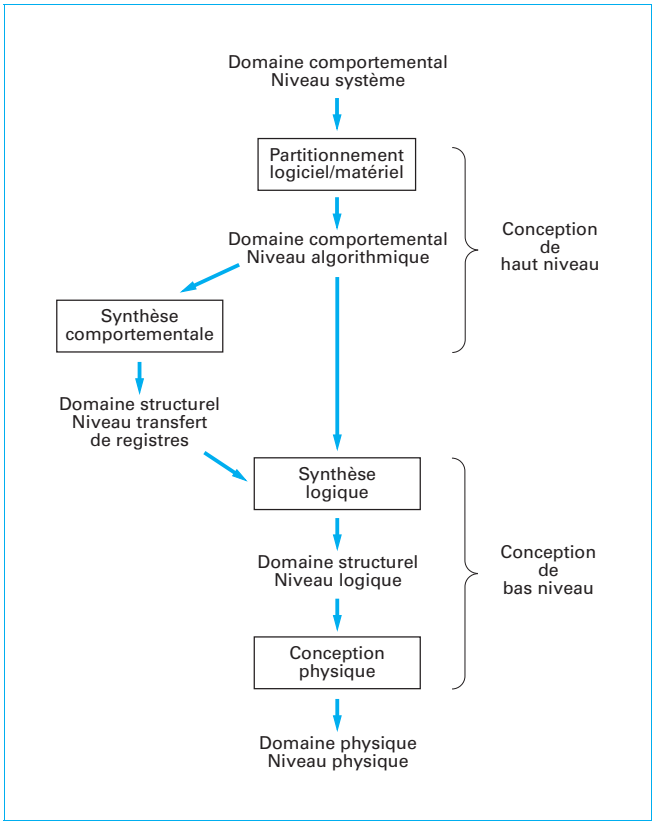


Figure 3 – Principales étapes de conception

Les **langages de description de matériel**, tels que VHDL [1] et Verilog [6] sont des langages qui supportent les concepts spécifiques au matériel et à sa conception. Ils décrivent plusieurs niveaux d'abstraction dans les domaines structurel et comportemental. Ils sont utilisés comme langage de simulation ou comme langage d'entrée des outils de synthèse automatique.

Certains langages de programmation (C, C++) ont été étendus pour accepter des concepts qui n'étaient pas à l'origine intégrés au langage, notamment les **notions de temps**, **de concurrence et de communication**, que l'on trouve dans les langages SystemC, SpecC, openCL [7], et de nombreux autres pour décrire le niveau système dans le domaine comportemental. Toujours dans ce domaine, on peut citer d'autres langages tels que les langages synchrones (Esterel, Lustre, Signal), et les langages et environnement de simulation (Matlab, SPW…). Au niveau structurel, des langages basés sur UML sont apparus, notamment IP-XACT.

Les systèmes monopuces, puis les systèmes multiprocesseurs, ont fait apparaître le besoin de représenter les parties matérielles et logicielles dans un **langage unique** dans un but de modélisation et/ou de validation fonctionnelle de l'ensemble du système. Les années 2000 ont vu la naissance des langages SystemC et SystemVerilog, qui permettent à la fois la description de matériel, mais aussi une utilisation plus informatique, pour représenter les parties logicielles applicatives (extension du langage C++ pour systemC, orientation objet pour systemVerilog). Cette tendance s'est poursuivie avec l'apparition du langage OpenCL, utilisé pour programmer les plateformes hétérogènes. Les outils de simulation se sont adaptés pour accepter ces langages et simuler un système plus complet. Les outils de synthèse (§ 3 et § 4) ont aussi évolué pour accepter comme point d'entrée ces langages.

2. Conception système

2.1 Intérêt

Pendant de nombreuses années, les chercheurs et les industriels se sont focalisés sur les problèmes de conception et de représentation aux niveaux physique, circuit et logique des éléments de base des circuits intégrés (transistors, portes logiques) qui constituent les plus bas niveaux d'abstraction. Les outils informatiques d'aide à la conception se sont développés conjointement (simulateurs logiques, outils de synthèse) depuis la fin des années 1980 et font maintenant partie intégrante du processus de conception de circuits.

Cependant, les applications sont devenues très complexes et les contraintes économiques sont telles que le temps de mise sur le marché est de plus en plus court, et la durée de vie d'un produit est aussi très limitée dans le temps. Les chercheurs et les industriels travaillent maintenant à un niveau d'abstraction plus élevé, appelé **niveaux systèmes**, pour s'affranchir des détails de réalisation, et ne conserver que des informations sur le codage du comportement. À un tel niveau d'abstraction, le concepteur peut traiter des applications plus complexes, et choisir la technologie qui réalise le mieux chacune des parties du système tout en respectant les contraintes sans se préoccuper de la description explicite du niveau physique des ressources utilisées. La conception est ainsi plus rapide, et la partie bas niveau est réalisée par des outils automatiques.

Il est aujourd'hui admis qu'une solution efficace pour concevoir un système consiste à développer une partie comprenant un ou plusieurs processeurs, appelée **partie logicielle**, **associée à une partie matérielle** composée de plusieurs circuits ou fonctions spécifiques (ASIC ou IP). Ce type d'architecture de système tire bénéfice de la facilité de programmation des processeurs et des performances des ASIC.

Mais la réalisation sur une même puce de ces systèmes implique non seulement de concevoir l'ASIC (ou l'acheter à un tiers), mais aussi de concevoir toute la partie logicielle, c'est-à-dire l'assemblage de processeurs, avec les périphériques nécessaires au bon fonctionnement (mémoire et hiérarchie mémoire, réseaux de communication, périphériques d'entrées/sorties, contrôleurs mémoire et d'interruptions…). On peut aussi s'appuyer sur des architectures existantes et les faire évoluer en fonction des besoins, mais il faut des ingénieurs spécialistes car la complexité est énorme.

La conception système brise le mythe de la conception séparée des parties logicielles et matérielles. En effet, ces différentes parties étaient généralement développées indépendamment l'une de l'autre puis assemblées. La séparation du développement conduisait à deux flots de conception indépendants, et les problèmes apparaissaient tardivement dans la phase d'intégration.

Le terme de **conception système** regroupe une méthode descendante de conception et des outils associés pour transformer la spécification d'un système en un produit satisfaisant les performances demandées, le coût et les contraintes de qualité et de temps de conception.

2.2 Description du flot usuel de conception système

La figure **4** illustre les différentes étapes de la conception d'un système. Il s'agit de pouvoir spécifier, valider, synthétiser (au sens de concevoir), simuler puis réaliser un prototype de l'application à concevoir.

Le point de départ est une **spécification du système à concevoir**. Cette spécification définit les propriétés du système et les contraintes. Cette spécification est généralement transformée pour obtenir une **spécification comportementale** validée par une simulation fonctionnelle.

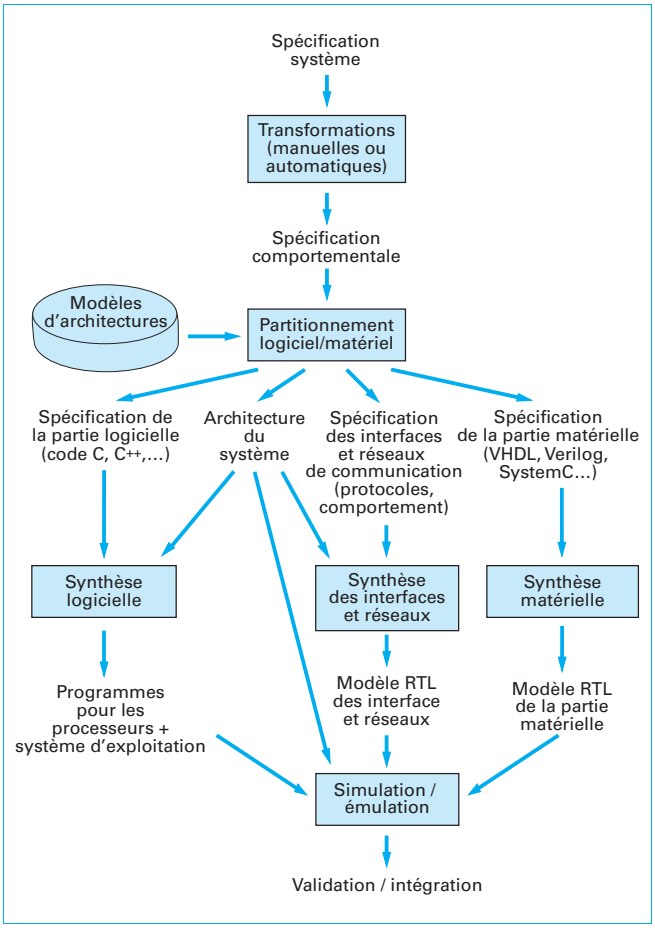


Figure 4 – Flot de conception système

À partir de cette spécification comportementale et d'un ensemble d'architectures possibles, on cherche la ou les architectures qui conviennent pour réaliser ce système. On obtient alors un **modèle d'architecture** du système. Cette architecture est un assemblage de composants. Il faut dans le même temps déterminer les parties du système qui seront réalisées en **logiciel** (programme s'exécutant sur un processeur) et celles qui le seront en **matériel** (composant spécifique). On obtient alors un ensemble de **spécifications** pour les parties logicielles, matérielles et les interfaces entre les différents composants de l'architecture.

Le support physique, qui réalise la fonctionnalité des différentes parties, est produit (automatiquement ou à la main). En fait, dans un premier temps, on se contente de produire un modèle permettant une simulation (dite aussi cosimulation) du système, même si on préfère depuis les années 2000, **l'émulation**. Dans le cas de la simulation, on parle de simulation bas niveau car très proche de la réalisation finale au niveau transistor. L'émulation consiste à produire une architecture physique très proche de l'architecture finale, sur un support configurable (FPGA par exemple) et permettant l'observabilité de signaux internes ou externes. On parle alors d'exécution (et non plus de simulation). On peut ainsi valider une partie du logiciel sur l'architecture matérielle. L'émulation de gros circuits se fait sur une plateforme multi-FPGA. Il s'agit alors de partitionner le circuit à émuler sur les multiples FPGA. Des liens

rapides permettent de connecter les différents FPGA. Des plateformes industrielles existent (Mentor Veloce 2 par exemple) avec un environnement pour l'aide au déploiement d'un circuit sur les FPGA et pour l'observation des signaux internes de l'architecture à valider.

La **synthèse logicielle** est la génération automatique d'un programme correct et efficace. Si plusieurs tâches concurrentes s'exécutent sur le même processeur, un système d'exploitation (SE) est nécessaire. La **synthèse matérielle** est la génération du circuit électrique par des outils de synthèse automatique ou par un concepteur. La **synthèse des interfaces** consiste à construire des mécanismes de communication entre les différentes parties du système. Ces interfaces sont soit en matériel (composants logiques), soit en logiciel (programmes ou pilotes), et mettent en œuvre des protocoles de communication fixés par le concepteur.

L'intérêt de la synthèse automatique réside dans l'augmentation de la productivité, ce qui diminue les coûts de développement et améliore la fiabilité du circuit généré.

Si la simulation ou l'émulation donnent des résultats convenables, la génération des composants et l'assemblage produisent un prototype, c'est l'étape d'**intégration**.

La synthèse logicielle n'étant pas le but de cet article, elle ne sera pas approfondie ici. La synthèse des interfaces et réseau est une étape difficile car le nombre de paramètres à prendre en considération est important (nature et volume des données, type de protocoles, bande passante, traitement des erreurs…). Cependant, il s'agit là aussi de produire soit du logiciel, soit du matériel, ce qui peut se ramener au flot de conception système. La synthèse matérielle comportementale et logique sera détaillée aux paragraphes 3 et 4.

2.3 Découpage logiciel/matériel

2.3.1 Nécessité et principe de base du découpage logiciel/matériel

Le découpage ou partitionnement logiciel/matériel (*hardware/software partitioning*) est une des phases principales de la conception de systèmes et consiste à rechercher le **meilleur compromis logiciel/matériel**. C'est dans cette phase que sont effectués les choix menant à une réalisation soit matérielle, soit logicielle des différentes parties constituant le système. En règle générale, le logiciel est utilisé pour réduire les coûts de conception et le matériel pour augmenter les performances. De nombreuses techniques et algorithmes ont été proposés pour aider le concepteur dans cette tâche, le but ultime étant bien sûr de l'automatiser.

La spécification comportementale étant découpée en sous-fonctions, le choix de leur réalisation, matériel ou logiciel, se pose. Mais comment choisir entre logiciel et matériel ? Les choix effectués dans cette étape de partitionnement sont des choix définitifs qui ne sont plus remis en cause dans les phases suivantes de synthèse. Un mauvais choix à ce niveau impose alors de recommencer le cycle de conception du partitionnement jusqu'à la simulation ou émulation. Il faut donc prêter une grande attention à cette étape.

Dans le contexte de la conception système, le logiciel peut être défini comme « **une séquence d'opérations s'exécutant sur une machine programmable nécessaire au fonctionnement d'un ensemble de traitements de l'information** ». Les machines programmables sur lesquelles est exécuté le logiciel sont de différents types : processeur général, processeur de traitement du signal (DSP), cœur de processeur…

Le **matériel** est défini par opposition au logiciel, comme étant « **une structure physique non programmable** ». Sa fonctionnalité est figée lors de la conception. Il n'est pas possible, comme pour le logiciel, d'effectuer une modification pour obtenir une nouvelle fonctionnalité. Un ASIC est assimilé à du matériel, car ce circuit spécifique est conçu pour une application donnée.

Il faut noter que certains composants sont à la frontière entre logiciel et matériel. Il s'agit des composants matériels logiques programmables tels que les FPGA (*Field Programmable Gate Array*). Les fonctions sont figées lors de la conception, mais la programmation des interconnexions modifie la fonctionnalité.

Les FPGA tiennent aujourd'hui une place importante dans un flot de conception. C'est tout d'abord une cible technologique pour la production d'un petit nombre de circuits. En effet, les FPGA offrent un rapport puissance de traitement/consommation énergétique très intéressant, d'une part et un coût de revient raisonnable, loin des coûts de fabrication des masques nécessaires à la production d'un circuit sur silicium, d'autre part. Mais c'est aussi une solution pour produire un modèle de l'architecture matérielle et ainsi tester le circuit ou système avant la production finale sur silicium. On parle alors de **prototypage**, ou d'**émulation**.

2.3.2 Algorithmes et outils de partitionnement

Cette étape dans la conception de systèmes consiste généralement à appliquer un algorithme d'optimisation sur l'ensemble des sous-fonctions de la spécification. On cherche alors à minimiser un critère (ou un ensemble de critères) donné tel que la surface, le temps d'exécution, la consommation ou autres. Cela suppose de maîtriser les problèmes d'estimation (ou d'évaluation) de ces critères pour une ou plusieurs architectures cibles. L'algorithme de partitionnement est donc un algorithme d'optimisation (ou de minimisation), cherchant une ou des réalisations optimisées pour un problème donné. Une excellente introduction à de telles heuristiques est effectuée dans l'article [RE 8].

Bien sûr, ce choix repose sur les caractéristiques des différents types de composants utilisés dans l'architecture cible visée. Le processeur standard offre la souplesse de programmation et le composant spécifique donne les temps de traitement les plus faibles. Il existe des solutions intermédiaires basées sur des cœurs de processeurs ou des processeurs spécialisés offrant des compromis souplesse-performance satisfaisants pour une large classe d'applications.

La fin des années 1990 a vu de nombreux travaux sur le partitionnement logiciel/matériel, que ce soit en termes de méthodes et d'outils ou d'algorithmes [5] [8]. Le nombre de publications relatif à ce domaine a beaucoup baissé sans que ce problème soit résolu. En fait, le problème est très complexe et dépend généralement de l'application et du moment. L'apparition d'un nouveau composant peut remettre en cause un partitionnement logiciel/matériel effectué quelques jours plus tôt.

De plus, les premiers travaux concernaient des architectures cibles relativement simples composées d'un processeur et de parties matérielles (ASIC), le tout étant interconnecté par un bus ou un réseau de communication simple. L'apparition, ces dernières années, d'applications complexes nécessite des architectures plus performantes : un seul processeur ne suffit plus. Il faut alors réaliser le système avec plusieurs processeurs, plusieurs circuits existants et un réseau de communication performant entre tous ces composants. Le partitionnement des différentes fonctionnalités sur une telle architecture devient beaucoup plus difficile, puisqu'il ne s'agit plus de choisir uniquement entre logiciel et matériel, mais sur quel logiciel et avec quel matériel.

3. Synthèse comportementale

La synthèse comportementale ou **synthèse de haut niveau** est un ensemble de transformations ou de raffinements appliqué à la spécification comportementale d'un circuit numérique synchrone pour

produire une description au niveau transfert de registres de ce circuit.

Des présentations plus approfondies de la conception comportementale sont données dans [4] [9] [10] [11].

3.1 Synthèse comportementale du matériel

La spécification d'entrée de la synthèse comportementale est une description sous forme algorithmique du circuit à concevoir. Il s'agit d'un ensemble d'instructions de calcul et de contrôle.

Les instructions de calcul (+, –…) définissent les calculs à réaliser. Les instructions de contrôle telles que les instructions conditionnelles ou les instructions itératives (*if…else*, *do…while*…) définissent le séquencement d'exécution des instructions de calcul. Cette spécification est écrite dans un langage algorithmique tel que le C ou le C++. Cette spécification est complétée par un ensemble de contraintes (surface, performance, exigences architecturales, interfaces des entrées-sorties, déroulage de boucles, pipelines…) que le circuit doit satisfaire. Ces contraintes peuvent être activées via des options de l'outil ou via des pragmas insérés dans la spécification.

La synthèse comportementale produit un circuit sous forme d'interconnexion de cellules combinatoires, de registres, de bus et de circuits de commande. Cette description est aussi généralement donnée en langage de description de matériel.

La synthèse comportementale consiste à sélectionner et à assembler astucieusement des composants issus d'une bibliothèque de composants décrits au niveau transfert de registres. L'objectif recherché est l'optimisation d'une fonction de coût (réduction de la surface, augmentation des performances…) qui guide la conception.

3.2 Architectures cibles et formes intermédiaires

Les deux modèles principalement utilisés sont la représentation interne de la description comportementale et l'architecture cible.

L'architecture cible fixe les performances du circuit. Une bonne maîtrise de cette architecture permet d'évaluer et de corriger les résultats de la synthèse comportementale.

La représentation interne ou format intermédiaire est une vue uniforme de la description comportementale, indépendante du langage de la description initiale, qui facilite les transformations. Un format intermédiaire est fait pour être manipulé par des outils, mais une bonne compréhension de sa sémantique permet au concepteur de comprendre le fonctionnement des outils de transformations ou de raffinement.

3.2.1 Architecture cible

L'architecture cible visée par la conception comportementale est divisée en deux parties : la **partie opérative** (appelée aussi chemin de données) et la **partie contrôle** (contrôleur) (figure **5**). La partie opérative reçoit des commandes de la partie contrôle. Elle exécute les opérations requises et rend compte des traitements. Elle traite des données entrées et produit des résultats. La partie contrôle peut recevoir des signaux externes (reset…).

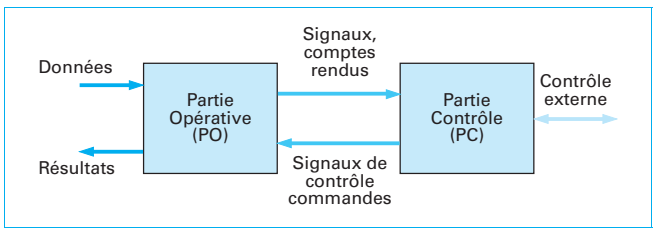


Figure 5 – Modèle d'architecture PC – PO

3.2.1.1 Partie opérative

La partie opérative est définie par sa structure interne et par son organisation fonctionnelle.

■ La **structure interne** décrit les composants utilisés et leur interconnexion. Chaque composant manipule des données et transforme, propage ou mémorise ces données. La partie opérative est composée d'un ensemble d'éléments : unités fonctionnelles, unités de mémorisation et unités de communication. Une unité fonctionnelle est un élément de la partie opérative qui exécute des opérations ou des transformations de données. Plus généralement, une unité fonctionnelle peut effectuer un ensemble d'opérations (logiques, arithmétiques…) dont l'exécution dépend de signaux de sélection. Chaque opération se caractérise par un temps d'exécution exprimé en nombre de cycles d'horloge. Les unités de stockage telles que les registres, bancs de registres ou mémoires enregistrent les données (variables ou constantes). Les unités de communication contrôlent les transferts de données entre les autres unités. Les unités de communication courantes sont les commutateurs (portes 3 états) qui permettent une connexion à un bus partagé, et les multiplexeurs qui commandent le transfert de données à partir de plusieurs entrées vers une seule sortie. Des unités de communication externe connectent la partie opérative au monde extérieur.

■ L'**organisation fonctionnelle** définit les transferts de données autorisés entre les composants de la partie opérative et les échanges avec la partie contrôle. Le modèle de transfert indique les dépendances entre la source et la destination d'un transfert, ainsi que les règles qui régissent ce transfert et les unités de communications à mettre en œuvre.

La figure **6** donne un exemple de partie opérative. Le résultat produit est le calcul de *f* à partir des valeurs (*a*, *b* et *d*) mémorisées. La sélection de l'opération de l'unité fonctionnelle, tout comme la commande des multiplexeurs, provient de la partie contrôle. L'horloge *clk* définit les points de synchronisation.

3.2.1.2 Partie contrôle

La partie contrôle gère les transferts dans la partie opérative en produisant les signaux de contrôle. Il existe deux modèles principaux pour réaliser un contrôleur : le **modèle de machines à états finis** et le **modèle microprogrammé**.

■ **Modèle de contrôleur basé sur une machine à états finis**

Un système séquentiel représenté par une machine à états finis se compose d'une partie combinatoire et d'une partie de mémorisation des états synchronisés dans le cas des systèmes synchrones.

Une machine à états finis (FSM pour « *Finite State Machine* ») est composée d'un ensemble d'états, d'un ensemble de transitions et d'un ensemble d'actions associé aux états (automates de Moore) ou aux transitions (automates de Mealy).

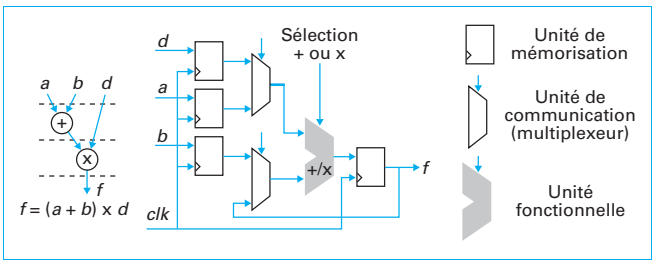


Figure 6 – Partie opérative associée au calcul de *f* = (*a* + *b*) x *d*

Plus formellement, une machine à états finis peut être décrite comme un quintuple (figure **7**) :

$$FSM = (S, I, O, FS, FO)$$

avec $S = \{s_i\}$ ensemble des états,

$I = \{i_j\}$ ensemble des ports d'entrée,

$O = \{o_k\}$ ensemble des ports de sortie,

$FS = \{fs \mid fs : S \times I \rightarrow S\}$ ensemble des fonctions de calcul de l'état suivant,

FO ensemble des fonctions qui mettent à jour les ports de sorties :

$$FO = \{fo \mid fo : S \rightarrow O\} \text{ pour l'automate de Moore}$$

$$FO = \{fo \mid fo : S \times I \rightarrow O\} \text{ pour l'automate de Mealy.}$$

Modèle de contrôleur microprogrammé

Le principe est d'utiliser une mémoire qui produit les signaux de contrôle **en sélectionnant le contenu d'une adresse** (figure **8**). Un séquenceur et les signaux de comptes rendus provenant de la partie opérative permettent d'obtenir l'adresse du prochain ensemble de signaux à envoyer à la partie opérative. Ce type d'architecture présente des avantages liés à la réutilisation et à la flexibilité de conception (des erreurs peuvent être corrigées plus facilement), mais il reste critiqué quant à ses performances (temporelles et de surface).

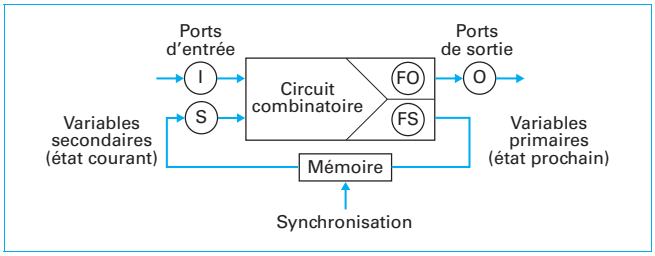


Figure 7 – Représentation d'un système séquentiel

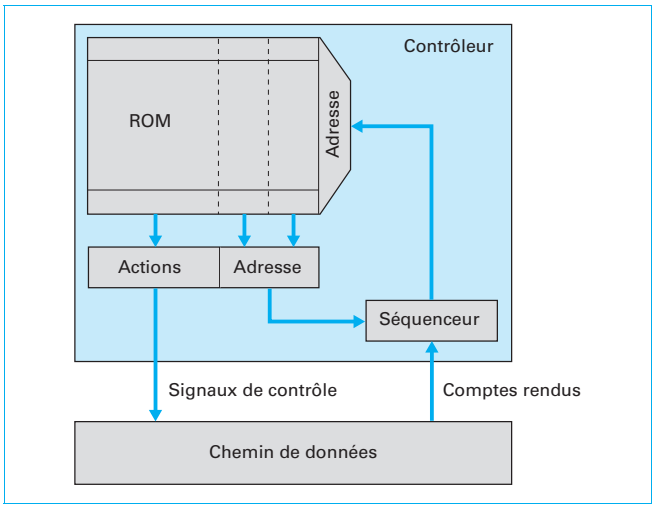


Figure 8 – Architecture d'un contrôleur microprogrammé

3.2.2 Formats intermédiaires

Les principaux formats intermédiaires sont basés sur des graphes. Le plus simple à comprendre est le **graphe de flot de données** (DFG pour *Data Flow Graph*), mais il existe aussi le **graphe de flot de contrôle** (CFG pour *Control Flow Graph*) et des **graphes mixtes** (CDFG pour *Control Data Flow Graph*).

Graphe de flot de données

C'est un graphe orienté $G = (V, E)$ où $V = \{v_1, \ldots, v_n\}$ est un ensemble fini de sommets, et $E \subset V \times V$ est une relation asymétrique de dépendance de données dont les éléments sont des arcs orientés.

Les sommets correspondent aux opérations. Un arc orienté e_{ij} de $v_i \in V$ à $v_j \in V$ existe, si la donnée produite par l'opération o_i (représentée par v_i) est consommée par l'opération o_j (représentée par v_j).

Exemple : sur la figure **9**, les quatre sommets représentent les quatre opérations (deux additions et deux multiplications). L'opération o_4 dépend des données produites par o_1 et o_2, et ne peut s'exécuter que quand les opérations o_1 et o_2 sont terminées. Les opérations o_3 et o_4 ne présentent aucune dépendance. Elles pourront être exécutées en même temps, ou aussi de façon séquentielle dans n'importe quel ordre.

Graphe de flot de contrôle

C'est un graphe orienté $G = (V, E)$ où $V = \{v_1, \ldots, v_n\}$ est un ensemble fini de sommets, et $E \in V \times V$ est une relation de flot de contrôle dont les éléments sont des arcs orientés de séquencement.

Les sommets correspondent à des instructions ou groupe d'instructions simples (affectation de variables, opérations arithmétiques et logiques, appels de procédures), à des instructions conditionnelles telles que *if*, *case*, *while* ou à des instructions de synchronisation (attente d'un événement extérieur).

Un arc orienté e_{ij} de $v_i \in V$ à $v_j \in V$ représente une relation de séquencement entre deux instructions. Un arc est étiqueté par une expression booléenne $cond_{ij}$ qui signifie que l'instruction v_j sera exécutée si l'instruction v_i est exécutée et si l'expression $cond_{ij}$ est vraie.

Exemple : la figure **10** extraite de [8] donne la description d'un double compteur. Les six sommets (notés v_1 à v_6) représentent les instructions de l'algorithme (donné en VHDL dans l'exemple). Les onze arcs représentent les relations de précédence entre ces instructions.

Graphes mixtes

Les spécifications comportementales contiennent généralement une combinaison d'instructions de calcul et d'instructions de séquencement. Les deux formats présentés précédemment ne suffisent pas pour des systèmes complexes. Il existe donc d'autres formats, basés sur une extension des graphes de flot de données et de contrôle, plus adaptés aux applications complexes.

Le modèle le plus classique est appelé **graphe de flot de contrôle et de données**. C'est une extension du graphe de flot de données, en ajoutant des sommets de contrôle pour les instructions de contrôle.

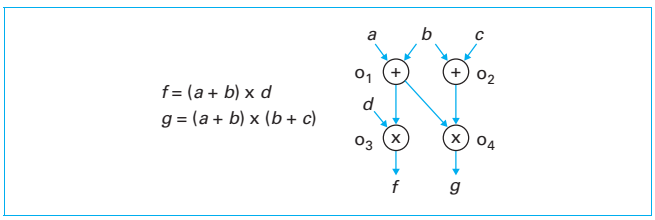


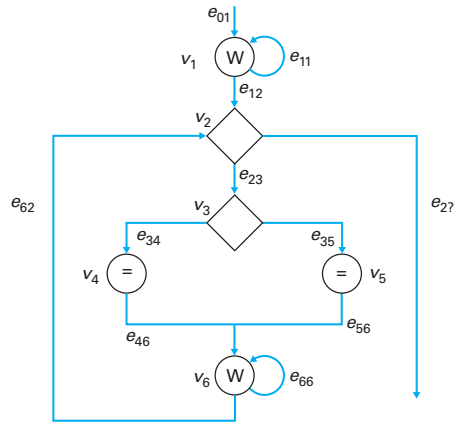
Figure 9 – Spécification comportementale et graphe de flot de données correspondant

3.3 Techniques de base

Il existe deux techniques de base utilisées en conception comportementale pour concevoir la partie opérative. L'**ordonnancement** fixe l'ordre d'exécution des traitements, et l'**allocation de ressources** associe des ressources physiques (éléments mémoire, opérateurs, multiplexeurs...) existantes pour réaliser l'architecture. Ces deux techniques sont appliquées sur le modèle de représentation interne.

3.3.1 Ordonnancement

La spécification comportementale est découpée en sous-graphes dont l'exécution peut être effectuée en un seul cycle de contrôle (pas de contrôle). À chaque cycle de contrôle, plusieurs opérations s'exécutent en parallèle si les ressources pour l'exécution sont présentes dans l'architecture. La première tâche de l'ordonnancement est donc d'extraire ces cycles de contrôle de la spécification. La deuxième tâche consiste à ordonnancer ces cycles d'exécution en respectant la spécification et les contraintes données. Il faut respecter la dépendance des données et de contrôle, inhérentes à la spécification initiale.




```
wait until start = '1' and rising_edge (clk)   (v1)
while start = '1' loop                         (v2)
   if sel = '1' then                           (v3)
      cnt_1 := cnt_1 + 1;                      (v4)
   else
      cnt_2 := cnt_2 + 1;                      (v5)
   end if;
   wait until rising_edge (clk);               (v6)
end loop;
```

Figure 10 – Spécification comportementale et graphe de flot de contrôle correspondant

De nombreux algorithmes d'ordonnancement ont été publiés. Nous nous limiterons dans cet article aux principes de base de l'ordonnancement appliqué aux graphes de flot de données.

L'ordonnancement d'un graphe de flot de données consiste à affecter chaque opération de spécification à un cycle de contrôle en respectant toutes les contraintes données, dont généralement le nombre de cycles et le nombre de ressources. Il s'agit alors d'un problème d'optimisation à deux dimensions : le temps et le nombre de ressources (donc la surface). On peut distinguer **quatre catégories d'algorithmes**.

Ordonnancement sans contrainte

L'ordonnancement **au plus tôt** (ASAP pour *As Soon As Possible*) affecte les opérations aux cycles de contrôle le plus tôt dès que les opérandes sont prêts. Au contraire, l'ordonnancement **au plus tard** (ALAP pour *As Late As Possible*) affecte les opérations aux cycles de contrôle le plus tard possible. Pour chaque opération, l'intervalle de temps obtenu avec ces deux algorithmes donne l'intervalle d'exécution (mobilité des opérations).

Ordonnancement sous contraintes de ressources

Des restrictions sur le nombre de ressources (notamment les unités fonctionnelles) sont imposées, par exemple pour limiter la taille du circuit. Les algorithmes d'ordonnancement par liste (*list scheduling*) sont efficaces pour résoudre ce problème.

Ordonnancement sous contraintes de temps

Dans ce cas, la spécification limite le nombre de cycles de contrôle, ce qui revient à borner le temps d'exécution. L'algorithme le plus connu dans le domaine est l'ordonnancement orienté par les forces (*force-directed scheduling*) [10].

Ordonnancement sous contraintes de temps et de ressources

C'est la combinaison des deux approches précédentes, résolue de façon optimale par des algorithmes de programmation en nombres entiers (*integer linear programming*).

Exemple : la figure **11** présente deux ordonnancements différents. L'ordonnancement en deux cycles est le résultat d'un ordonnancement ASAP. L'architecture pour effectuer en même temps les deux additions, puis les deux multiplications nécessite la présence de deux opérateurs d'addition et deux opérateurs de multiplication. L'ordonnancement en trois cycles ne requiert qu'un seul opérateur d'addition et un seul opérateur de multiplication. C'est une des solutions les plus rapides sous contraintes de ressources (un additionneur et un multiplieur).

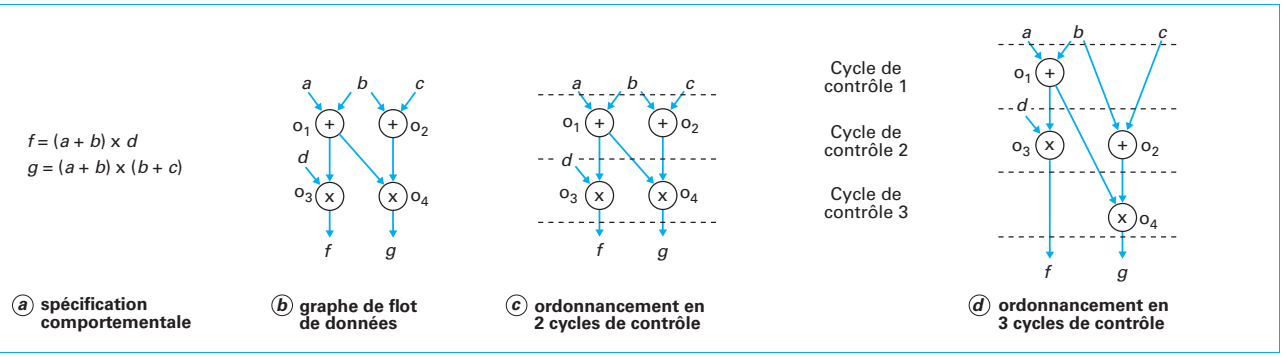


Figure 11 – Exemples d'ordonnancements d'un graphe de flot de données

3.3.2 Allocation de ressources

L'allocation de ressources détermine le nombre et le type des ressources utilisées dans le circuit, c'est-à-dire le nombre d'unités **fonctionnelles**, d'unités de **mémorisation**, et d'unités de **communication**. La quantité de ressources limite le parallélisme de la partie opérative et influence aussi l'ordonnancement.

■ L'**allocation de ressources** consiste à choisir dans une bibliothèque les composants les plus adaptés pour réaliser les opérations ou instructions de la spécification. Par exemple, concernant les unités fonctionnelles, une opération d'addition peut être réalisée par un opérateur d'addition, par une unité arithmétique et logique, ou par un coprocesseur complexe.

■ L'**allocation des registres** (unités de mémorisation) consiste à déterminer le type et le nombre de registres nécessaires pour stocker les données et les résultats de calculs intermédiaires. Il faut aussi déterminer l'affectation des variables aux registres, c'est-à-dire décider dans quel registre est placée chacune des variables.

■ L'allocation des **unités de communication** définit la façon de transporter les données entre les unités fonctionnelles et les éléments de mémorisation. Principalement, deux types d'architectures s'opposent :

– les **architectures à base de multiplexeurs** sont introduits entre les registres et les unités fonctionnelles. À chaque cycle de contrôle, les multiplexeurs sont commandés par la partie contrôle pour transférer les variables vers les opérateurs. De même, l'entrée des registres peut nécessiter des multiplexeurs si un registre contient, à des cycles différents, des variables provenant d'opérateurs différents. L'optimisation associée est la minimisation du nombre de multiplexeurs, obtenue en permutant l'ordre d'entrée des variables pour les opérations commutatives ;

– les **architectures à base de bus** partagent un ou plusieurs bus auxquels sont connectés les registres et les unités fonctionnelles. Le nombre maximal de bus est obtenu en calculant le nombre maximal de transferts de données dans un même cycle de contrôle. Des cellules trois états pourraient être ajoutées si les sources de données partagent un bus, ou si les destinations de données sont reliées à plus d'un bus. Cependant, les cellules trois états peuvent induire des court-circuits à la mise sous tension, et on préfère l'utilisation de multiplexeurs. D'ailleurs, ces cellules trois états présentes au sein des FPGA ont rapidement été supprimées.

Exemple : la figure **12** illustre tout le principe d'allocation de ressources. Le point de départ est un graphe de flot de données ordonnancé (figure **12*a***). La première étape consiste à choisir les unités fonctionnelles pour réaliser les opérations d'addition et de multiplication. La solution retenue comporte deux unités arithmétiques et logiques (ALU1 et ALU2) capables de réaliser les deux opérations. Il a aussi été décidé dans cet exemple que l'ALU1 réaliserait les opérations o_1 et o_3, tandis que ALU2 réaliserait les opérations o_2 et o_4. L'allocation des registres a été faite en plaçant les valeurs *a*, *b*, *c*, *d*, *e* dans cinq registres. Pour ne pas ajouter de registres supplémentaires, les calculs intermédiaires *x* et *y* obtenus par les additions sont aussi placés dans les registres des valeurs qui ne sont plus nécessaires à partir de ce cycle de contrôle. Donc *x* est placée dans le registre r1 où était mémorisé *a*, qui ne sera plus utilisé ensuite. Il en est de même pour *y*. Les valeurs des résultats f_1 et f_2 sont aussi placées dans les registres r1 et r2. On connecte ensuite l'entrée des opérateurs aux registres à travers des multiplexeurs ou des bus.

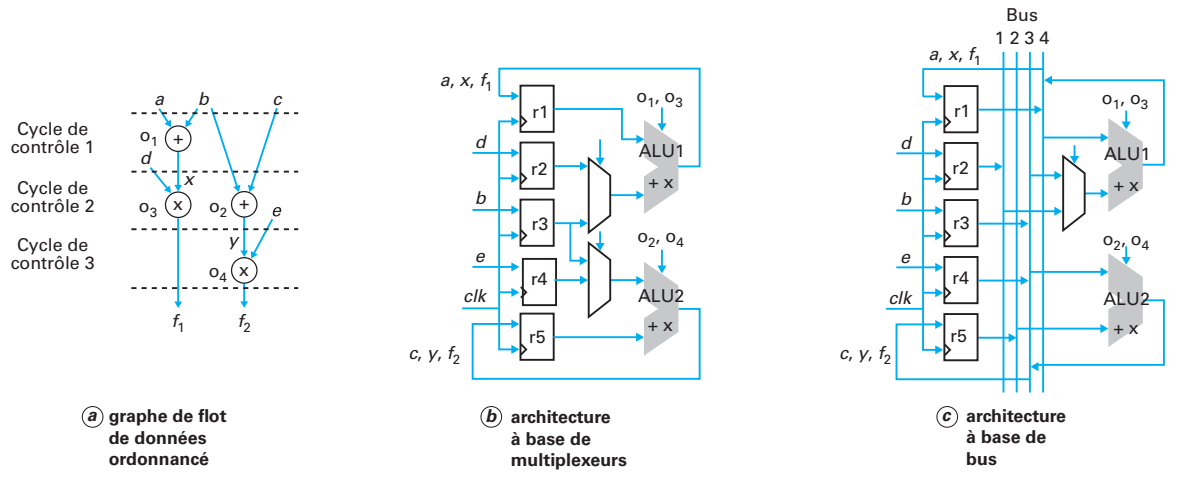
3.4 Flot de synthèse comportementale

Le flot de synthèse comportementale comporte quatre étapes principales : la **compilation**, les **transformations de la représentation interne**, l'**ordonnancement** et l'**allocation de ressources**.

La spécification comportementale est donnée généralement sous une forme textuelle dans un langage de programmation tel que le langage C ou des langages de description de matériel. Pour faciliter la suite du processus de conception, une représentation interne est nécessaire. Dans l'étape de **compilation**, le programme écrit en langage source est traduit dans une représentation interne.

La **représentation interne** obtenue ne permet pas de générer une architecture efficace, car elle contient des structures proches du langage source. En appliquant quelques transformations sur la représentation initiale, on peut obtenir un modèle plus simple. Par exemple, la propagation des constantes autorise le remplacement d'une opération effectuée sur des valeurs constantes par le résultat en supprimant l'opération. On élimine aussi les opérations redondantes (plusieurs calculs effectués sur les mêmes variables). L'élimination du code mort supprime les parties de la représentation interne qui n'interviennent pas dans le calcul de la valeur finale. Il s'agit toujours d'optimisations dans le but de simplifier et de rendre plus performant le circuit conçu.

Ensuite, les étapes d'**ordonnancement** et d'**allocation** (et d'affection) **de ressources** s'enchaînent dans un ordre quelconque. En fait,



ⓐ graphe de flot de données ordonnancé

ⓑ architecture à base de multiplexeurs

ⓒ architecture à base de bus

Figure 12 – Exemples d'architectures à multiplexeurs et à bus partagés

ces étapes sont interdépendantes mais, pour des raisons de complexité, la plupart des outils les exécutent indépendamment, traitant le plus souvent l'ordonnancement avant l'allocation de ressources. Ces étapes peuvent être guidées dans les outils modernes par l'utilisation de directives utilisateurs (pragmas ou options). Ces directives permettent notamment de forcer l'architecture cible à respecter un degré de parallélisme sur une boucle ou à l'implanter sous forme de pipeline. En modifiant ces directives, on peut alors influer sur la structure de l'architecture (parallélisme, pipeline) pour respecter des contraintes de débit ou de latence par exemple.

À l'issu de ces étapes, on obtient une description de circuit sous forme d'interconnexion de cellules combinatoires, de registres, de bus et de circuits de commande. Ce modèle dit de niveau RTL (*Register Transfer Level*) correspond à l'entrée des outils de synthèse logique.

3.5 Outils de synthèse comportementale

La première génération d'outils de synthèse comportementale date du début des années 1980. Il s'agissait de prototypes issus des laboratoires de recherche. Jusque vers le milieu des années 2000, la qualité du code générée par les outils n'a guère convaincu les industriels. NEC avec son outil de synthèse comportementale CyberWorkbench faisait office d'exception durant cette période. Depuis une dizaine d'années, beaucoup d'entreprises ont sauté le pas de la synthèse comportementale grâce aux progrès réalisés par la dernière génération d'outils, aussi bien académiques (AUGH, GAUT, LEGUP…) qu'industriels (Catapult, VivadoHLS…). Les outils égalent maintenant en qualité les solutions codées manuellement dans un langage RTL. Cet essor s'appuie sur de nombreux ajouts de fonctionnalités par rapport à la génération d'outils précédentes, que nous allons traiter brièvement.

3.5.1 Gestion des interfaces de communications

Initialement, les outils de synthèse comportementale se sont cantonnés à supporter des interfaces d'entrée-sorties simples, type FIFO ou protocole poignée de main. L'intégration d'interface plus complexe était laissée à la charge de l'utilisateur qui devait faire le lien entre l'interface de l'outil et son besoin. Les outils actuels sont capables de libérer l'utilisateur de ce travail en lui fournissant par exemple des interfaces de communications de type bus AXI, PCI Express ou accès générique à une mémoire centrale partagée. L'abstraction de ces interfaces se fait généralement via les directives des outils, ce qui facilitent la portabilité du code.

3.5.2 Gestion des opérateurs dédiés des FPGAs

Les FPGAs contiennent des opérateurs dédiés : blocs de mémoire embarqués, blocs de calcul DSP, blocs de calcul flottant. Ces blocs ont été introduits dans les FPGA, car les implémentations classiques étaient trop gourmandes en ressources. Leur reconnaissance implicite par les outils de synthèse comportementale est une garantie de performance et de portabilité.

3.5.3 Langages et restrictions

Les restrictions imposées par les outils sur les langages ont longtemps été un frein. Certains outils supportent maintenant plusieurs langages en entrée (C,C++, SystemC) et de sortie (Verilog, VHDL). Les restrictions sur les styles de codages se réduisent : il est maintenant courant de pouvoir utiliser des types de données arbitraires (entier, point fixe ou flottant) à la taille désirée. Ceci est rendu possible par l'intégration automatique d'IPs issues des bibliothèques de l'outil. Cette fonctionnalité offre un grand pas en avant pour la réutilisation du matériel.

3.5.4 Directives de contrôle de l'architecture

Les directives de contrôle de l'architecture (pragma ou options) sont certainement celles qui ont joué le rôle le plus important dans l'adoption des outils de HLS. Avec elle, l'utilisateur peut imposer assez précisément à l'outil le type de transformation de boucles voulues en apportant très peu de changement dans le code source. L'utilisateur peut renseigner l'outil sur l'indépendance d'itérations de boucles facilitant le travail à l'outil de synthèse HLS pour extraire du parallélisme. Il est par exemple possible de choisir d'implanter un pipeline et/ou des déroulements de boucles, de gérer les conditions limites de manière statique ou dynamique, ou encore de gérer la cadence d'un pipeline. Ces directives ne se limitent pas qu'à la gestion des boucles : il est aussi courant d'en trouver permettant de découper les mémoires pour faire apparaître du parallélisme exploitable par l'outil. Dans tous les cas, elles permettent de guider l'outil vers de meilleures solutions sans avoir à se soucier des modifications complexes imposées à l'architecture.

3.5.5 Intégration dans le flot de conception

La nouvelle génération d'outils de HLS facilite grandement le travail d'intégration dans l'écosystème global de conception. Par exemple, les estimateurs de performance et de surface utilisés sont notamment beaucoup plus fiables, ce qui permet au concepteur de réduire considérablement le nombre d'itérations nécessaires à effectuer avec les outils en aval. Auparavant, des sorties trop optimistes de l'outil de HLS empêchaient régulièrement les outils de synthèse logique de fonctionner correctement, ce qui obligeait l'utilisateur de l'outil de HLS à prendre une marge supplémentaire. Une autre avancée considérable est la génération automatique d'environnement de vérification : à partir de bancs de tests décrits dans le langage d'entrée, l'outil génère l'équivalent dans le langage de sortie et compare automatiquement le bon fonctionnement sur les stimuli fournis. Cela offre à l'utilisateur un point de vérification fiable et rapide avant d'attaquer la suite du flot de conception. Enfin, on peut aussi évoquer les fonctionnalités d'automatisation en amont dans le flot de conception. Le choix des directives est un travail d'exploration d'architecture que certains outils ont automatisé pour accélérer le temps de conception.

4. Synthèse logique et conception physique

La synthèse logique et la conception physique constituent la partie **bas niveau** (*back-end*) de la conception d'un système par opposition à la conception de **haut niveau** (*front-end*).

La **synthèse logique** est une étape de conception qui transforme une description RTL en un réseau de portes logiques (figure **13**). Cette étape est maintenant réalisée automatiquement à l'aide d'outils de synthèse commerciaux très efficaces.

La **conception physique** correspond aux étapes ultimes de la conception des circuits intégrés (figure **13**). Les résultats représentent l'implémentation pour obtenir le dessin des masques utiles à la fabrication du circuit. Les trois étapes principales sont le placement, le routage et la vérification du dessin physique (*layout*). Ces étapes de conception sont automatisées depuis les années 1980.

Les principes de base de la synthèse logique et de la conception physique sont développés dans [12].

4.1 Synthèse logique

La synthèse logique regroupe deux étapes qui sont effectuées séquentiellement : l'**optimisation logique** et la **conversion** (ou projection) **technologique** (*technology mapping*).

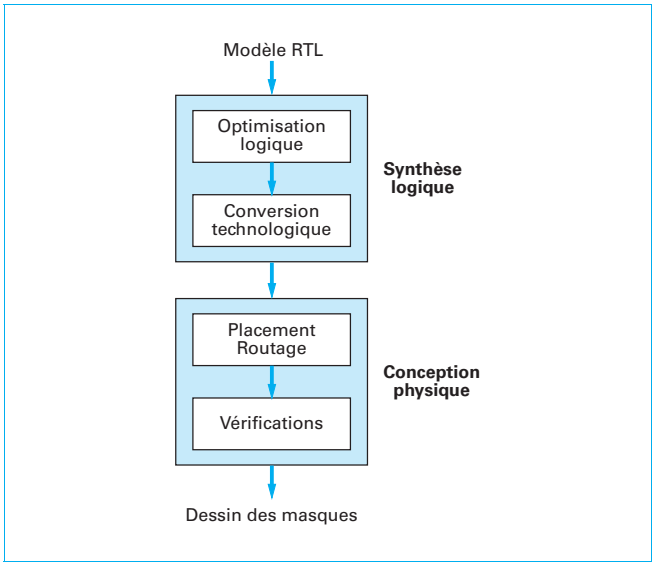


Figure 13 – Flot de conception logique et physique

L'**optimisation logique** est appliquée pour réduire la complexité du modèle RTL. On cherche alors à limiter la redondance des calculs, tout en préservant la fonctionnalité du circuit. On utilise des techniques d'optimisation booléennes telles que l'algorithme de Quine/ MacCluskey développé dans les années 1950 ou d'autres versions améliorées qui réduisent le nombre de littéraux (un littéral est l'apparition d'une variable dans une équation) des équations booléennes, donc le nombre de portes logiques réalisant cette fonction.

Exemple : la fonction f_1 définie de la manière suivante :

$$f_1 = abcd + a\bar{b}cd + abc\bar{d} + a\bar{b}c\bar{d} + ade + bc + bde$$

devient, après son optimisation :

$$f_1 = ac + ade + bc + bde$$

Dans des cas simples, comme celui présenté ci-dessus, quelques transformations logiques ou l'utilisation de tableaux de Karnaugh produisent une solution optimisée. Pour des équations de plusieurs centaines de littéraux, des outils d'aide sont nécessaires.

La factorisation est aussi une technique d'optimisation qui consiste à rechercher les expressions ou sous-expressions communes à plusieurs fonctions pour réduire aussi la taille du circuit réalisant les fonctions.

Exemple : la figure **14** présente un exemple de factorisation d'équations booléennes.

La **conversion technologique** consiste à décomposer les expressions booléennes et à les « réécrire » en utilisant des cellules disponibles dans une bibliothèque (figure **15**). Un outil de synthèse logique utilise une bibliothèque de cellules prédéfinies, telles que des portes logiques simples (ET, OU…) à plusieurs entrées, ou des cellules à plusieurs étages (composées de portes logiques simples en cascade). Cette bibliothèque est fournie par le fabricant du circuit, et elle constitue la cible technologique.

La synthèse logique produit un schéma sous forme de portes logiques, ainsi qu'un schéma en transistors des fonctions logiques.

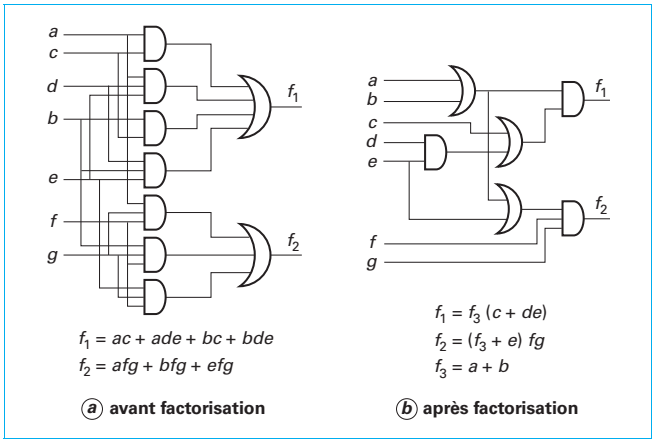


Figure 14 – Exemple de factorisation d'équations booléennes

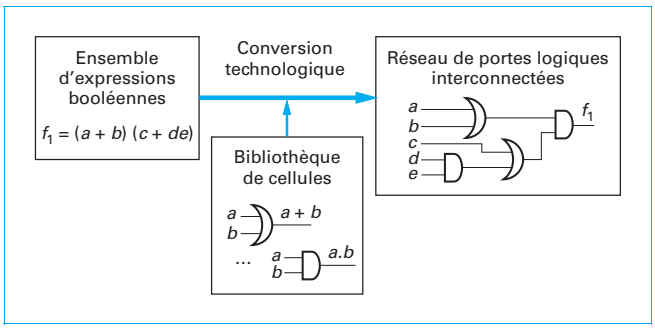


Figure 15 – Conversion technologique

4.2 Conception physique

Le schéma en transistors obtenu par la synthèse logique doit maintenant être transformé pour obtenir les masques du circuit. Un circuit intégré se définit par son dessin des masques. Les masques sont représentés par une juxtaposition de polygones issus des dessins des transistors (ou des cellules) et des interconnexions. Ils sont utilisés pour la fabrication des circuits intégrés.

L'étape de placement fixe la position des différents éléments du circuit dans le plan. L'étape de routage connecte ces éléments. Ces deux étapes peuvent être déterminantes quant aux performances finales du circuit. Un mauvais placement entraîne de longues interconnexions et des délais de propagation qui ralentissent le fonctionnement du circuit. La distribution des horloges est une des difficultés liées au placement, car le déphasage d'une horloge entraîne généralement une mauvaise prise en compte des évènements et induit des dysfonctionnements. À noter qu'il existe des circuits sans horloge, appelés circuits asynchrones.

La simulation, après placement et routage, permet de valider ces deux étapes. La difficulté est l'extraction des caractéristiques (capacités parasites, temps de propagation…) liées à la technologie et au circuit pour réaliser une simulation la plus proche possible du circuit final.

Il reste ensuite à effectuer plusieurs vérifications pour s'assurer que les règles de dessin (la technologie de fabrication impose des contraintes géométriques qui définissent alors les règles de dessin) sont respectées (DRC pour *Design Rule Check*), que les règles électriques ne sont pas violées (ERC pour *Electrical Rule Check*) et que la fonctionnalité après placement routage est bien identique à celle de départ (LVS pour *Layout Versus Schematic*).

5. Tendances et évolutions récentes

5.1 Évolution vers la conception des circuits multiprocesseurs monopuces

La complexité des applications dans les domaines du multimédia, de la téléphonie mobile ou des jeux est telle que les systèmes qui réalisent ces fonctions sont composés de plusieurs processeurs (ou nœuds ou cœurs de calcul) associés à des composants existants interconnectés par un réseau de communication performant (figure **16**). À cela s'ajoute un espace mémoire pour stocker un volume de données très important, puisque, dans certaines applications, la moitié de la surface du système est due à la mémoire. Étant donné que tous ces systèmes correspondent à des marchés de masse, ils sont tous (ou seront) intégrés sur une seule puce (SoC pour *System on Chip* ou système monopuce) afin de réduire les coûts de production. Ces systèmes sont les principaux vecteurs d'orientation de toute l'industrie des semiconducteurs.

L'architecture de ces systèmes multiprocesseurs monopuces est décomposée en couches (figure **17**) pour maîtriser la complexité des parties matérielle et logicielle. La couche la plus basse de la partie matérielle contient les principaux composants de l'architecture. La couche du réseau de communication est un ensemble de dispositifs qui permet aux différents composants d'interagir. Il s'agit maintenant d'un réseau de communication complexe. Les couches supérieures concernent la partie logicielle. La couche d'interface logiciel/matériel isole le matériel (auquel elle est intimement liée) du reste du logiciel à l'aide de fonctions de bas niveau permettant d'accéder aux ressources matérielles. Les programmes de l'application (couche haute) sont adaptés à l'architecture par un ensemble de fonctions de gestion de ressources (partage du temps du processeur, de la mémoire…), généralement réalisées par le système d'exploitation.

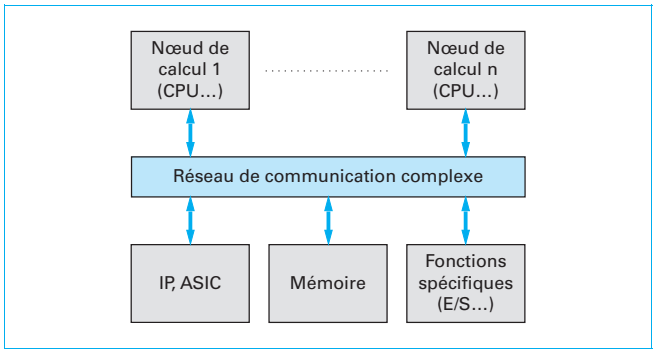


Figure 16 – Architecture générique physique d'un système multiprocesseur monopuce

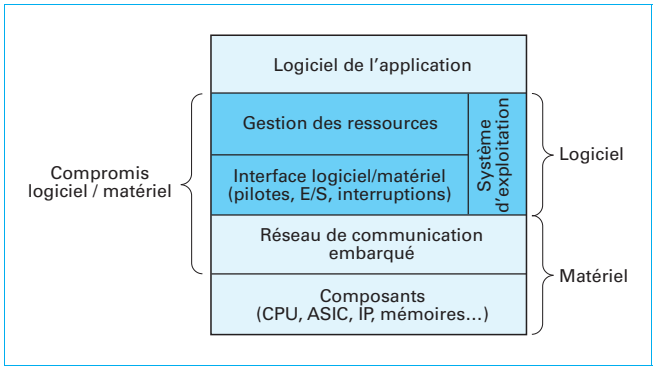


Figure 17 – Architecture des systèmes multiprocesseurs monopuces

Un tel modèle d'architecture induit des changements dans la méthode de conception.

Au niveau de l'**architecture matérielle**, les composants de base sont séparés de la couche de communication. Dans les schémas traditionnels, le travail de l'architecte consistait à tailler des composants sur mesure, afin de prévoir les interconnexions de manière efficace et d'obtenir les meilleures performances. Ce schéma n'est plus applicable à partir d'un certain degré de complexité. Aussi, pour les systèmes monopuces, le travail de l'architecte consiste à assembler des composants existants en vue de respecter des contraintes de performances et de coûts au niveau du système global et non plus au niveau du seul composant. Ainsi, la valeur ajoutée de la conception des systèmes monopuces se situe plutôt au niveau de la couche de communication.

Au niveau de l'**architecture logicielle**, la complexité des applications impose une décomposition du code en couches et l'utilisation d'un système d'exploitation. Ainsi, on maintient la séparation des métiers pour le développement du logiciel et du matériel. La partie la plus basse du logiciel correspond à la couche HAL (*Hardware Abstraction Layer*) qui représente l'interface avec les parties matérielles pour l'accès aux composants, généralement à travers l'accès à des registres.

Au niveau de l'**architecture globale**, les choix des composants de base et du modèle de communication ne peuvent plus être réalisés sans prendre en compte l'organisation du logiciel. Les compromis logiciel / matériel fixent les grands choix d'architecture, ainsi que les limites entre les couches de communication matérielles et les couches basses de communication logicielles. En effet, l'existence de facilités matérielles pour les entrées / sorties simplifie la couche de communication logicielle.

5.2 Réseaux sur puce

Les circuits actuels avec des fréquences de fonctionnement de l'ordre du GHz et des technologies nanométriques (une dizaine de nanomètres en 2018) ont des temps de propagation des signaux, à travers le circuit, supérieurs à la période de l'horloge. Ceci entraîne des incertitudes sur les retards des signaux. La synchronisation du système avec une seule horloge est impossible. On parle de **systèmes globalement asynchrones et localement synchrones**.

La recherche d'une faible consommation de puissance reste vraie, notamment pour des applications embarquées fonctionnant sur batterie. Ces considérations énergétiques poussent les concepteurs vers des technologies faible tension d'alimentation (de l'ordre de 1 V), dans lesquelles le bruit électrique, les interférences électromagnétiques et les injections de charges produisent des erreurs sur les données ou des erreurs supplémentaires sur les retards. Avec toutes ces sources d'erreurs, la transmission des informations est peu fiable.

Les réseaux de communication deviennent alors le point clé dans la conception de tels systèmes qui exigent donc des architectures de communication flexibles et parallèles. Un réseau sur puce (NoCs pour *Network-on-Chip*) est une infrastructure de communication qui facilite l'interconnexion de composants à grande échelle. Un NoC apparaît comme la technologie la plus adaptée pour la communication de SoCs modernes, car ils ont une plus grande flexibilité et offrent un parallélisme par rapport à d'autres solutions telles que les connexions point à point et les bus de communication. Les NoCs possèdent un vaste espace de conception concernant différents aspects comme la topologie, le routage, le contrôle de flux et la qualité de services entre autres.

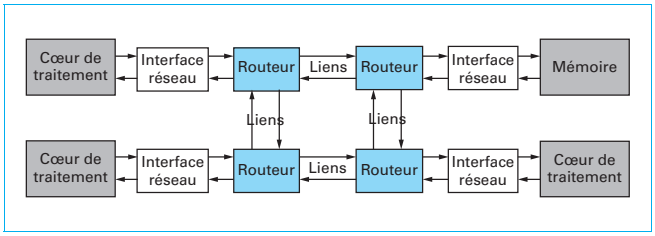


Figure 18 – Représentation d'un système basé sur un réseau sur puce 2 x 2

Un réseau sur puce est une **architecture simple à quatre composants principaux** : les cœurs de traitement, les interfaces réseau (IR), les routeurs et les liens. La figure **18** représente l'architecture d'un NoC pour un maillage 2 x 2. Les cœurs de traitement sont les clients des services de communication du NoC. Ils utilisent le réseau pour transmettre et recevoir des informations. Ces cœurs sont multiples, mais sont généralement des processeurs (classiques ou spécialisés), des blocs mémoire ou des circuits spécialisés. Les IR sont responsables de l'interface entre les cœurs et le réseau, et également responsables de la communication de bout en bout du réseau. Les routeurs sont utilisés pour transmettre des paquets à travers le réseau, et pour fournir des mécanismes de contrôle de flux. Ils sont reliés aux IR sur les ports locaux. Enfin, les liens sont les fils physiques interconnectant les ports des routeurs créant une topologie du réseau.

Dans la littérature consacrée au sujet, le terme réseau sur puce est utilisé à la fois pour parler du système complet et pour parler de l'infrastructure de communication.

Les systèmes multiprocesseurs commercialisés en 2018 sont basés sur des réseaux sur puce, comme par exemple les processeurs Kalray (MMPA) ou Tilera (Tile64). Il en est de même de certains projets « *open source* » comme le projet « *open core* ».

5.3 FPGA comme cible technologique

Comme indiqué paragraphe 4, la cible technologique classique d'un circuit intégré est un processus d'intégration de transistors. En suivant ce processus, le fondeur peut intégrer dans le silicium le circuit décrit par l'intermédiaire d'un jeu de masques. Néanmoins, la durée de ce processus d'intégration se mesure en mois, ce qui impacte très négativement le temps de mise sur le marché. De plus, les coûts fixes de ce type de processus sont très importants. On parle en millions d'euros pour des technologies récentes. De ce fait, seules les grandes séries sont concernées pour rester économiquement viables.

Pour réduire les coûts et le délai de mise sur le marché, le concepteur de circuit peut cibler les circuits logiques programmables, parmi lesquels les FPGA [H 1 196] sont les plus puissants. Ces circuits intégrés peuvent être configurés après fabrication de manière à accueillir n'importe quel circuit logique (§ 4.1) dans la limite de ses ressources. Ces dernières sont des éléments de logique combinatoire (des tables de vérités, appelées *Look Up Table*-LUT-) et de logique séquentiel (bascules D), et sont assemblées via une matrice d'interconnexion. En comparaison avec une cible technologique classique, les circuits logiques programmables sont, à circuit logique égal, plus lents, plus gourmands en énergie et aussi plus chers à l'unité. Leur immense avantage est d'être disponible sur étagère et de permettre l'utilisation du circuit immédiatement après sa configuration. Ils sont donc naturellement utilisés pour le prototypage et pour l'émulation de circuits. Pour ces cibles technologiques, le flot de conception varie très légèrement du flot classique (figure **12**). Le but n'est plus de créer des dessins de masques, mais de générer une configuration sous la forme d'un flux binaire, appelé *bitstream* en anglais. Pour y arriver, le flot réalise une synthèse logique comme dans le cas classique. La projection technologique est réalisée sur les éléments constitutifs du circuit logique programmable. L'utilisation de LUT accélère grandement cette étape. En se basant sur la structure interne du circuit logique programmable, l'optimisation du circuit est obtenue également par des algorithmes de placement-routage. Contrairement au cas classique, le placement n'est pas libre, mais contraint par la position prédéfinie des ressources. On cherche ensuite à minimiser les chemins via le routage. Les algorithmes de ce problème, connu sous le nom de *timing closure*, sont donc différents de ceux du cas classique. Après placement-routage, le circuit logique est complètement défini. Chaque élément du circuit programmable peut être configuré et l'ensemble de ces configurations peut être assemblé pour former le *bitstream*.

5.4 Marché des IP

La mise sur le marché rapide (*time-to-market*) des systèmes intégrés supposent de concevoir puis de produire très rapidement les circuits. Une solution efficace est de reprendre une partie ou des fonctionnalités qui ont déjà donné lieu à une réalisation. On parle d'IP (*Intellectual Properties*) pour nommer ces circuits ou fonctionnalités, disponibles chez de nombreux fournisseurs, dont quelques français (Design & Reuse, Accelize…). La connexion entre et avec les IP est facilitée grâce à des standards de communication (OCP-IP, bus ARM). Des outils de qualification d'IP sont aussi utilisés pour valider leur intégration et le respect des standards.

6. Conclusion

Les systèmes numériques nécessaires pour les applications les plus performantes sont constitués de circuits intégrés comportant quelques milliards de transistors. La conception de ces circuits est un vrai défi et ne peut se faire qu'avec l'aide d'outils qui automatisent une grande partie des étapes de conception. Cet article a présenté l'ensemble des étapes de conception des circuits numériques : modèles, langages, et étapes automatisées de synthèses comportementale, logique et physique.

Les outils d'aide à la conception vont bien sûr continuer à évoluer en fonction de la demande pour produire des circuits et systèmes plus gros et plus complexes. On constate toutefois un intérêt toujours plus grand pour l'utilisation des FPGA comme cible technologique, que ce soit pour le prototypage ou pour le produit fini. Ceci ne change pas fondamentalement les étapes de conception. Une bonne intégration des outils pourrait permettre à des non-spécialistes d'utiliser les FPGA comme accélérateurs de traitement, et ainsi accroître la diffusion et l'intérêt pour la conception de circuit. Les industriels majeurs du FPGA (Xilinx et Intel Altera) mettent beaucoup d'efforts pour aller dans ce sens.

7. Glossaire

ASIC (*Application Specific Integrated Circuit*) :
Circuit intégré spécifique pour une application.

CAD (*Computer Aided Design*) :
Acronyme anglais de CAO.

CAO :
Conception Assistée par Ordinateur.

CPU (*Computing Unit*) :
Microprocesseur.

description comportementale :

Description d'un système par un ensemble d'échanges d'informations (par opposition à une description structurelle).

HAL (*Hardware Abstraction Layer*) :

Couche d'abstraction du matériel, interface de programmation qui permet d'accéder aux composants matériels.

IP (*Intellectual Property*) :

Circuit spécialisé.

FPGA (*Field Programmable Gate Array*) :

Circuit composé d'un ensemble de cellules ou blocs électroniques dont les interconnexions sont programmables.

MPSoC (*Multi-Processor System-on-Chip*) :

Système multiprocesseur sur puce.

placement :

Définition de la position des cellules et blocs à la surface d'une puce.

routage :

Définition des interconnexions à l'intérieur d'une puce.

SoC (*System on Chip*) :

Système monopuce.

description structurelle :

Description d'un système par un ensemble d'interconnexions entre les composants (par opposition à une description comportementale).

VHDL (*Very high speed integrated circuit Hardware Description Language*) :

Langage de description de matériel.

Verilog :

Langage de description de matériel.

VLSI (*Very Large Scale Integration*) :

Circuit ou système intégré contenant au minimum quelques millions de transistors.

Conception des systèmes VLSI

par **Frédéric ROUSSEAU**
Professeur des universités
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), TIMA, Grenoble, France

et **Olivier MULLER**
Maître de conférences des universités
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), TIMA, Grenoble, France

Sources bibliographiques

[1] AIRIAU (R.), BERGE (J.M.), OLIVE (V.) et ROUILLARD (J.). – *VHDL : langage, modélisation, synthèse*. Presses Polytechniques et Universitaires Romandes (1998).

[2] HUBNER (M.) et BECKER (J.). – *Multiprocessor System-on-Chip*. Springer (2011).

[3] DE MICHELI (G.) et BENINI (L.). – *Networks on Chips*. Morgan Kaufmann (2006).

[4] NIEMANN (R.). – *Hardware/sofware Co-design for Data Flow Dominated Embedded Systems*. Kluwer Academic Publishers (1998).

[5] GAJSKI (D.D.). – *High Level Synthesis : Introduction to Chip and System Design*. Kluwer Academic Publishers (1992).

[6] THOMAS (D.E.) et MOORBY (P.). – *The VERILOG Hardware Description Language*. Kluwer Academic Publishers (1991).

[7] GAJSKI (D.D.), ZHU (J.), ZOMER (R.), GERSTLAUER (A.) et ZHAO (S.). – *SpecC Specification Language and Methodology*. Kluwer Academic Publishers (1997).

[8] JERRAYA (A.A.). – *Conception de haut niveau des systèmes monopuces*. Hermes (2002).

[9] JERRAYA (A.A.). – *Behavioral Synthesis and Component reuse with VHDL*. Kluwer Academic Publishers (1997).

[10] PAULIN (P.G.) et KNIGHT (J.P.). – *Force Directed Scheduling for the Behavioral Synthesis of ASIC's*. IEEE transactions on CAD, 6, pp. 661-679 (1989).

[11] VALKERT (R.A.) et CAMPOSANO (R.). – *A survey of High Level Synthesis Systems*. Kluwer Academic Publishers (1994).

[12] JERRAYA (A.A.). – *Conception logique et physique des systèmes monopuces*. Hermes (2002).

[13] GAJSKI (D.D.) et VAHID (F.). – *Specification and Design of Embedded Hardware-Software Systems*. IEEE Design & Test of Computers, pp. 53-67, Printemps (1995).

À lire également dans nos bases

AUGUIN (M.), VERDIER (F.) et AFFES (H.). – *Techniques de gestion de la puissance dans les systèmes sur puce*. [H 8 270] (2016).

ÉTIEMBLE (D.). – *Introduction aux systèmes embarqués, enfouis et mobiles*. [H 8 000] (2016).

ÉTIEMBLE (D.). – *Évolution de l'architecture des ordinateurs*. [H 1 058] (2016).

ENCRENAZ-TIPHENE (E.). – *Méthodes formelles pour la vérification des systèmes embarqués*. [H 8 250] (2013).

GAMATIE (A.) et TORRES (L.). – *Introduction à la conception conjointe matériel/logiciel – Une vision générale*. [H 8 450] (2017).

HOUZET (D.). – *Calcul généraliste sur carte graphique – Du rendu au calcul massivement parallèle*. [TE 5 990] (2016).

LOUIS (S.) et FARAHMAND (B.). – *L'essor des objets connectés : Introduction*. [H 1 509] (2015).

PETROT (F.). – *OS embarqués*. [H 8 200] (2011).

SENTIEYS (O.) et TISSERAND (A.). – *Architecture reconfigurable FPGA*. [H 1 196] (2012).

SIARRY (P.). – *Application des métaheuristiques d'optimisation en électronique, traité recherche*. [RE 8] (2002).

Sites Internet

Marché du semiconducteurs :
https://www.wsts.org/ (Word Semiconductor Trade Statistics)
Feuille de route des semiconducteurs :
https://irds.ieee.org/ (International Roadmap for Devices and Systems)
Société Kalray :
http://www.kalrayinc.com/
Société Tilera, maintenant Mallanox :
http://www.tilera.com/ et https://mellanox.com/
Forum MPSoC :
http://www.mpsoc-forum.org/
Laboratoire IM2NP :
http://www.im2np.fr/
CMP – CNRS :
https://mycmp.fr/
Design and Reuse :
https://www.design-reuse.com/

Cadence :
https://www.cadence.com/
Synopsys :
https://www.synopsys.com/
Xilinx :
https://www.xilinx.com/
Altera :
https://www.altera.com/ puis https://www.intel.com
Mentor Graphics :
https://www.mentor.com/

Accelize :
https://www.accelize.com/
Laboratoire TIMA :
http://tima.univ-grenoble-alpes.fr/tima/fr/index.html
LIP6 :
https://www.lip6.fr/
Lab-STIC :
https://www.labsticc.fr/

Événements

MPSoC forum : Forum international sur les systèmes multiprocesseurs sur puce – Logiciel dédié au matériel :
http://www.mpsoc-forum.org/
Conférence DAC – Design Automation Conference :
http://www.dac.com/

Conférence DATE – Design Automation & Test in Europe :
http://www.date-conference.com/
ESWeek – Embedded System Week :
http://www.esweek.org/

TECHNIQUES DE L'INGÉNIEUR

GAGNEZ DU TEMPS ET SÉCURISEZ VOS PROJETS EN UTILISANT UNE SOURCE ACTUALISÉE ET FIABLE

Techniques de l'Ingénieur propose la plus importante collection documentaire technique et scientifique en français !

Grâce à vos droits d'accès, retrouvez l'ensemble des **articles et fiches pratiques de votre offre, leurs compléments et mises à jour,** et bénéficiez des **services inclus.**

RÉDIGÉE ET VALIDÉE PAR DES EXPERTS	MISE À JOUR PERMANENTE	100 % COMPATIBLE SUR TOUS SUPPORTS NUMÉRIQUES	SERVICES INCLUS DANS CHAQUE OFFRE

- **+ de 350 000 utilisateurs**
- **+ de 10 000 articles de référence**
- **+ de 80 offres**
- **15 domaines d'expertise**

- **Automatique - Robotique**
- **Biomédical - Pharma**
- **Construction et travaux publics**
- **Électronique - Photonique**
- **Énergies**
- **Environnement - Sécurité**
- **Génie industriel**
- **Ingénierie des transports**
- **Innovation**
- **Matériaux**
- **Mécanique**
- **Mesures - Analyses**
- **Procédés chimie - Bio - Agro**
- **Sciences fondamentales**
- **Technologies de l'information**

Pour des offres toujours plus adaptées à votre métier, découvrez les offres dédiées à votre secteur d'activité

Depuis plus de 70 ans, Techniques de l'Ingénieur est la source d'informations de référence des bureaux d'études, de la R&D et de l'innovation.

www.techniques-ingenieur.fr

CONTACT : Tél. : + 33 (0)1 53 35 20 20 - Fax : +33 (0)1 53 26 79 18 - E-mail : infos.clients@teching.com

TECHNIQUES DE L'INGÉNIEUR

LES AVANTAGES ET SERVICES compris dans les offres Techniques de l'Ingénieur

ACCÈS

Accès illimité aux articles en HTML
Enrichis et mis à jour pendant toute la durée de la souscription

Téléchargement des articles au format PDF
Pour un usage en toute liberté

Consultation sur tous les supports numériques
Des contenus optimisés pour ordinateurs, tablettes et mobiles

SERVICES ET OUTILS PRATIQUES

Questions aux experts*
Les meilleurs experts techniques et scientifiques vous répondent

Articles Découverte
La possibilité de consulter des articles en dehors de votre offre

Dictionnaire technique multilingue
45 000 termes en français, anglais, espagnol et allemand

Archives
Technologies anciennes et versions antérieures des articles

Impression à la demande
Commandez les éditions papier de vos ressources documentaires

Alertes actualisations
Recevez par email toutes les nouveautés de vos ressources documentaires

*Questions aux experts est un service réservé aux entreprises, non proposé dans les offres écoles, universités ou pour tout autre organisme de formation.

ILS NOUS FONT CONFIANCE

AgroParisTech · AIRBUS · cea · CESI Enseignement Supérieur et Formation Professionnelle · cnes Centre National d'Études Spatiales · CSTB le futur en construction · ENGIE · GE · L'ORÉAL · Leroy-Somer · École Polytechnique Université Paris-Saclay · SAFRAN Aerospace Defence Security · Sagemcom · SAINT-GOBAIN · Schneider Electric · THALES · TOTAL · Valeo · VEOLIA · Wallonie · SPW Service public de Wallonie

www.techniques-ingenieur.fr

CONTACT : Tél. : + 33 (0)1 53 35 20 20 - Fax : +33 (0)1 53 26 79 18 - E-mail : infos.clients@teching.com