# AAAA : asynchronisme et adéquation algorithme architecture

# Asynchronism in a Joint Algorithm Architecture Perspective

**par Marc RENAUDIN, Frédéric ROBIN*, Pascal VIVET**

ENST de Bretagne-Antenne de Grenoble,
* France Telecom / CNET-Grenoble
CNET / DTM / CET
BP 98, Chemin du Vieux Chêne
F-38243 MEYLAN
e-mail : marc.renaudin@cnet.francetelecom.fr

***résumé et mots clés***

**Bien que leurs techniques de conception gagnent en maturité avec notamment l'apparition d'outils automatiques, et que le nombre des réalisations augmente, les circuits asynchrones constituent aujourd'hui encore une curiosité. Toutefois, Philips dans les pagers et Sharp-Mitsubishi dans les Set Top Box sont les premiers à intégrer des circuits asynchrones dans des équipements pré-industriels. Cet article présente dans un premier temps le principe de fonctionnement des circuits asynchrones en le situant par rapport à celui des systèmes synchrones. Les propriétés et potentiels généraux ainsi que les coûts d'implémentation inhérents à ce type de réalisation sont ensuite abordés. Ces deux premières parties introductives constituent les bases du dernier chapitre de l'article qui s'attache à montrer que la combinaison du mode de fonctionnement asynchrone et d'un formalisme de description basé sur les processus séquentiels communicants constitue un outil puissant pour l'étude et la conception conjointe des algorithmes et des architectures. Nous introduisons en particulier la notion de convergence algorithme-architecture qui traduit l'idée qu'algorithme et architecture peuvent être confondus en un seul modèle, décrit sous la forme d'un programme unique qui rend compte à la fois des propriétés fonctionnelles et structurelles de la spécification.**

**Asynchronisme, circuits VLSI, adéquation algorithme-architecture.**

*abstract and key words*

Even if the design of asynchronous circuits reaches a certain level of maturity, they are still considered as a curiosity. However, some interesting industrial applications have been developed recently by Philips for pagers and by Sharp-Mitsubishi for set-top boxes. In this paper, the operating mode of asynchronous circuits is first presented, and compared with its synchronous counter-part. The properties and potentials of asynchronous circuits are then described as an introduction to the last part of the paper which aims at showing that the asynchronous operating mode associated with a description formalism based on communicating processes is a powerful tool to jointly design algorithms and architectures.

Asynchronism, VLSI circuits.

## 1. introduction

La conception de la plupart des circuits intégrés logiques est facilitée par deux hypothèses fondamentales : les signaux manipulés sont binaires, et le temps est discrétisé. La binarisation des signaux permet une implémentation électrique simple et offre un cadre de conception maîtrisé grâce à l'algèbre de Boole. La discrétisation du temps permet quant à elle de s'affranchir des problèmes délicats de rétroactions et de transitoires.

Si les circuits asynchrones conservent un codage discret pour les signaux, et la plupart du temps binaire, ils se distinguent des circuits synchrones car ils sont régis par un mode de fonctionnement moins contraint. Cet article tente de montrer que par l'adoption

d'un mode de fonctionnement moins contraint, les circuits asynchrones apparaissent comme une famille très attractive de circuits qui offre des potentiels très importants dans le domaine du développement conjoint d'algorithmes et d'architectures.

Nous préciserons tout d'abord le mode de fonctionnement des circuits asynchrones et dans quelles mesures il est moins contraint. Cela permettra de clairement les positionner vis-à-vis du mode de fonctionnement synchrone. Nous décrirons ensuite les concepts de communication et de contrôle qui constituent les fondements du mode de fonctionnement asynchrone. Nous analyserons dans le cas général les potentiels qu'il offre dans le contexte de la conception de systèmes de traitement de l'information. Nous pourrons enfin tenter de convaincre le lecteur que ce mode de fonctionnement ouvre des perspectives nouvelles pour la réalisation de systèmes et qu'il favorise l'étude conjointe des algorithmes et des architectures.

# 2. vous avez dit « Asynchrone » ?

## 2.1. le mode de fonctionnement asynchrone

Ce paragraphe est destiné à clarifier l'utilisation du terme asynchrone dans le contexte de cet article. Il vise également à introduire le mode de fonctionnement asynchrone et à exhiber ses différences avec le mode de fonctionnement synchrone.

« Asynchrone » signifie qu'il n'existe pas de relation temporelle *a priori* entre les objets auxquels il se rapporte. Dans le contexte de la conception des systèmes, ces objets sont la plupart du temps des événements au sens large (contrôle ou données). On a parfois besoin de traiter des signaux asynchrones dans les systèmes. C'est par exemple le cas d'un signal d'interruption appliqué à un microprocesseur. En général on s'efforce de limiter ces situations car elles sont source d'indéterminisme, mal maîtrisé avec des réalisations synchrones. Nous avons montré qu'en fait l'indéterminisme généré par l'asynchronisme entre signaux peut conduire à des algorithmes et implémentations plus performants [1]. Les circuits dits « asynchrones » n'ont pas vocation à traiter des signaux asynchrones, mais comme les circuits synchrones ils traitent des signaux dont la causalité des occurrences est parfaitement connue *a priori*. Toutefois, nous verrons dans la suite que ce type de circuits permet de traiter les signaux asynchrones avec plus d'efficacité et de fiabilité que les circuits synchrones. Qualifier ces systèmes d'asynchrones est donc abusif et ambigu. La différence entre systèmes asynchrones et synchrones ne se situe donc pas dans la nature de la relation temporelle qui existe entre les signaux. Où se situe réellement la différence?

Il faut aller chercher la réponse dans le mode de fonctionnement de ces systèmes. Les systèmes dits « asynchrones » fonctionnent avec la seule connaissance de l'ordre d'occurrence des événements, comme les systèmes flot de données. Le système est spécifié en décrivant l'enchaînement des événements et des opérations sous la forme d'un graphe de dépendances (réseau de Pétri par exemple). L'évolution globale est garantie par l'évolution conjointe et possiblement concurrente des différents éléments qui composent le système. Chaque élément évolue avec les seules « informations » reçues des éléments avec lesquels il est « connecté ». On entend par « informations reçues » des variables ou des événements de synchronisation. Les règles d'activation sont similaires à celles des réseaux de Pétri (une place est activée si toutes les entrées sont marquées; l'exécution a lieu si toutes les sorties ne sont pas marquées; après exécution les marques sont supprimées des entrées et les sorties sont marquées). Il n'existe pas de mécanisme global d'activation du système (l'analogie est aussi très forte avec le modèle des processus séquentiels communicants [2]). Et c'est là que se situe la différence avec les systèmes synchrones pour lesquels le signal d'horloge joue le rôle d'un actionneur global. Tous les éléments du système évoluent ensemble lors de l'occurrence d'un événement horloge, l'exécution de tous les éléments est donc synchronisée. Ce mécanisme global d'activation introduit une contrainte de synchronisation sur le début et aussi la fin de toutes les actions. Tous les éléments doivent respecter un temps d'exécution maximum fixé par la fréquence des occurrences des événements d'activation (condition de bon fonctionnement). L'activité de tous les éléments du système est ainsi synchronisée dans le temps.

A l'opposé, les systèmes asynchrones évoluent de façon localement synchronisée et le déclenchement des actions dépend uniquement de la présence des données à traiter. La correction fonctionnelle est indépendante de la durée d'exécution des éléments du système. C'est pourquoi nous pensons que la meilleure façon de qualifier ce type de systèmes est : « *localement synchronisés* ». Ainsi, lorsque « systèmes ou circuits asynchrones » apparaît dans le texte, il faut comprendre « systèmes ou circuits localement synchronisés ».

## 2.2. des opérateurs aux caractéristiques plus complexes

D'un point de vue externe un opérateur asynchrone peut être considéré comme une cellule réalisant une certaine fonction et communiquant avec son environnement à travers des canaux de communication banalisés. Le point clé est que ces canaux de communication servent à échanger des données avec l'environnement aussi bien que des informations de synchronisation (parfois les deux simultanément) et ceci en utilisant un protocole prédéfini (*cf.* § 2.3.a). L'opérateur peut indifféremment réaliser une fonction au niveau bit, au niveau arithmétique ou même implémenter un algorithme complexe, il peut être combinatoire ou à mémoire, mais il doit toujours respecter la même sémantique de communication.

En plus du protocole de communication utilisé, un opérateur asynchrone peut être caractérisé par trois paramètres fondamentaux.

– Un temps de latence, qui correspond à la chaîne combinatoire la plus longue nécessaire à une sortie pour être l'image fonctionnelle d'une entrée. Comme nous le verrons dans la suite, le temps de latence peut être variable en fonction des données d'entrée. Ses variations dépendent de l'algorithme et de l'implémentation choisis. Il conviendra donc pour être plus précis de définir pour chaque opérateur un temps de latence minimum, moyen et maximum (pire cas) [5] [31] [32] [39].

– Le temps de cycle, qui caractérise le temps minimum qui sépare l'acceptation de deux informations en entrée. C'est dans le cas général, le temps nécessaire pour échanger une donnée entre deux ressources de mémorisation connectées. Il est au minimum égal au temps de cycle de l'étage d'entrée et au maximum égal au temps de cycle de l'étage le plus lent de l'opérateur. Ainsi, le temps de cycle vu de l'entrée dépend de l'occupation des étages internes de l'opérateur.

– La profondeur du pipeline de l'opérateur définit le nombre maximum de données ou d'informations que l'opérateur peut mémoriser. La même définition est utilisée pour caractériser des opérateurs synchrones à mémoire avec la différence que dans le cas asynchrone le nombre de données présentes dans le pipeline n'est pas imposé. C'est pourquoi on définit une profondeur de pipeline maximale.

On peut remarquer que les opérateurs asynchrones sont caractérisés par les mêmes paramètres que les opérateurs synchrones, à la différence que ces paramètres peuvent varier dynamiquement, c'est à dire en cours d'exécution ou de fonctionnement du circuit, et ceci en préservant la correction fonctionnelle. Le mode de fonctionnement des circuits asynchrones apparaît donc plus riche, car il va permettre des finesses d'exécution inaccessibles au mode de fonctionnement synchrone. C'est la complexité du protocole de communication, et la puissance de la synchronisation locale qui permettent d'exploiter toutes les finesses d'exécution. On peut reformuler cela en disant que l'approche synchrone « pire cas » permet de concevoir un système en manipulant des vues externes simplistes voire réductrices des opérateurs. Ces vues peuvent se réduire à une bande passante et un taux de pipeline. Il n'est pas nécessaire de considérer les schémas d'exécution internes qui sortent du cadre de la « chaine critique ».

Est-ce un avantage? Est-ce un inconvénient? Cela dépend de quel point de vue on se place. Au niveau de la réalisation nous donnerons quelques éléments généraux, mais ce n'est pas l'objet de cet article. Par contre, nous aborderons plus en détails le point de vue Adéquation Algorithme Architecture. Sans tirer de conclusion hâtive, on peut d'ores et déjà entrevoir que le spectre des architectures sera élargi par la prise en compte de ces nouveaux paramètres. La partie suivante fera le lien entre ces paramètres et les potentiels fonctionnels qu'on peut en attendre. On peut déjà dire que la spécification des opérateurs devra être plus fine, et en conséquence les connaissances sur leur fonctionnement et les algorithmes approfondies.

## 2.3. le principe de base : un contrôle local

Comme nous venons de le voir le point fondamental du mode de fonctionnement asynchrone est que le transfert d'information est géré localement par une signalisation adéquate. Les opérateurs connectés se synchronisent en échangeant des informations indépendamment des autres opérateurs auxquels ils ne sont pas connectés. Ce mécanisme garantit la causalité des événements au niveau local et donc la correction fonctionnelle du système. Le contrôle local doit en conséquence remplir les fonctions suivantes : être à l'écoute des communications entrantes, déclencher le traitement localement si toutes les informations sont disponibles et produire des valeurs sur les sorties. Cependant, afin d'être en mesure d'émettre de nouvelles données, les opérateurs doivent être informés que les données qu'ils émettent sont bien consommées. Sans acquittement comment savoir que le récepteur est prêt à traiter une nouvelle donnée? Pour permettre un fonctionnement correct, le contrôle local doit prendre en charge cette signalisation bidirectionnelle. Toute action de communication doit être acquittée par le récepteur afin que l'émetteur puisse émettre à nouveau. Les communications sont dites à poignée de mains ou de type requête-acquittement.

*a) Les protocoles de communications*

Pour implémenter une telle gestion des échanges, deux principaux protocoles de communication sont utilisés : le protocole 2 phases, figure 2.1 (ou NRZ pour Non Retour à Zéro ou encore « Half-handshake »), et le protocole 4 phases, figure 2.2 (ou RZ pour Retour à Zéro ou encore « Full-handshake »). Il faut noter que dans les deux cas tout changement d'un signal est acquitté par un autre signal. C'est ce qui permet d'assurer l'insensibilité aux temps de traitement. Un changement des données est acquitté par le signal d'acquittement et un changement du signal d'acquittement est acquitté par un changement des données et ainsi de suite.

Il est difficile de dire *a priori* lequel de ces protocoles est le meilleur. Le protocole quatre phases requiert deux fois plus de transitions que le protocole deux phases. Il est *a priori* plus lent

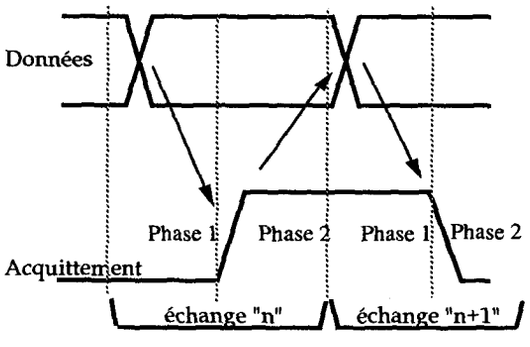



**Figure 2.1. – Protocole de communication 2 phases.**

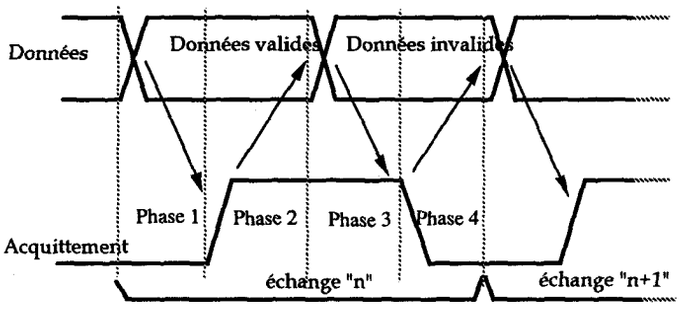



**Figure 2.2. – Protocole de communication 4 phases.**

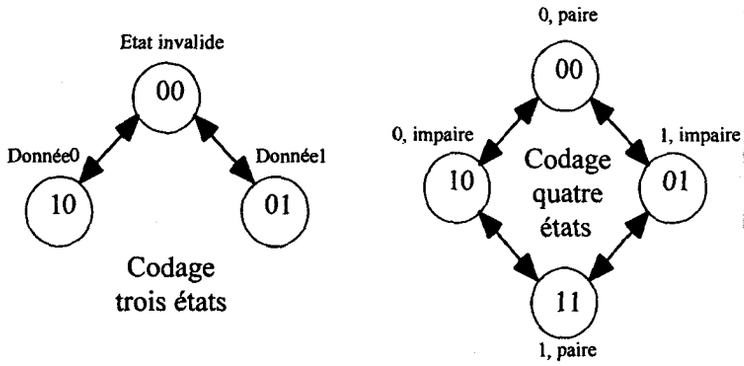



**Figure 2.3. – Codage trois et quatre états des données.**

et consomme plus d'énergie. Toutefois, dans la plupart des applications le temps de calcul domine le temps de communication entre opérateurs. Il est alors possible d'effectuer des traitements en parallèle des deux phases de retour à zéro.

En ce qui concerne la consommation il faut étudier l'implémentation de ces protocoles. Le protocole deux phases nécessite un matériel plus important que le protocole quatre phases car il doit détecter des transitions et non pas des niveaux. Le nombre plus faible de transitions utilisé est souvent compensé par la complexité du matériel, et la consommation est comparable à celle des réalisations quatre phases.

*b) Le codage des données*

Une question subsiste : comment détecter la présence d'une donnée et comment générer un signal qui indique la fin d'un traitement ? La réponse est dans l'adoption d'un codage particulier pour les données. Il est en effet impossible d'utiliser un seul fil par bit de donnée. Un fil par bit ne permet pas de détecter que la nouvelle donnée prend un état identique à la précédente. Les protocoles utilisent communément un codage bifilaire ou double rail pour chaque bit de donnée. Cela double donc le nombre de fils par rapport aux réalisations synchrones ou aux réalisations basées sur un protocole données groupées (*Cf.* Figure 2.4). Avec deux fils par bit de donnée, quatre états sont utilisables pour exprimer deux valeurs logiques (« 0 », « 1 »). Deux codages sont communément adoptés : l'un utilisant trois états seulement et l'autre utilisant les quatre états.

Pour le codage trois états (cf. Figure 2.3) , un fil prend la valeur 1 pour coder une donnée à 1 et l'autre fil prend la valeur 1 pour coder une donnée à 0. L'état « 11 » est interdit alors que l'état « 00 » représente l'invalidité d'une donnée (utile pour le protocole 4 phases *Cf.* Figure 2.2). Il est alors aisé de réaliser une circuiterie qui détecte les passages de « 00 » à « 01 » ou « 10 » (validité d'une donnée ou fin de traitement) et les passages de « 01 » ou « 10 » à « 00 » (invalidité d'une donnée).

La convention adoptée dans le cas du codage quatre états, est de coder les valeurs 0 et 1 d'un bit avec deux combinaisons. L'une des combinaisons est considérée comme étant de parité impaire et l'autre de parité paire (*Cf.* Figure 2.3). Chaque fois qu'une donnée

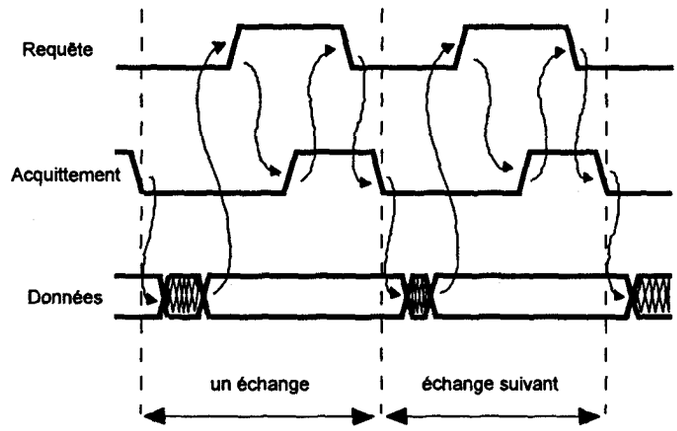



**Figure 2.4. – Protocole de communication 4 phases «données groupées» («Bundled Data»).**

est émise, on change sa phase. Ceci permet de passer d'une valeur logique à une autre sans passer par l'état d'invalidité. L'analyse de la parité permet de détecter la présence d'une nouvelle donnée et de générer un signal de fin de calcul. Cette méthode très élégante est cependant d'une implémentation très coûteuse [3].

Il est clair que le codage trois états est bien adapté au protocole 4 phases alors que le codage quatre états est bien adapté au protocole 2 phases. L'implémentation du codage quatre états est plus complexe, plus lente et consomme plus que celle du codage trois états. En effet, le codage quatre états nécessite d'implémenter une petite machine à états dans chaque cellule. Cette machine à états, outre la mémorisation de l'état courant, assure la génération du code et la détection des changements de phase. Plusieurs logiques ont été proposées pour implémenter le codage 3 états dans les opérateurs de traitement. Une comparaison de ces logiques est proposée dans [4] et [5]. [5] propose également une étude approfondie de la « logique cascode différentielle » qui constitue l'implémentation la plus efficace du codage trois états.

Dans le but de simplifier la détection de présence d'une donnée multibit, il a été proposé de séparer l'information de contrôle de l'information de donnée proprement dite. On crée explicitement

un signal de contrôle unique, souvent dénommé « requête », qui s'apparente à un signal d'horloge local. Il est utilisé pour déclencher la mémorisation ou le traitement des données qui lui sont associées. L'avantage principal est que les bits de données peuvent être codés à l'aide d'un seul fil. Cette technique est connue sous le nom de « Bundled Data » ou données groupées (Figure 2.4) [6]. Il est possible de faire de même pour l'acquittement d'une donnée multibit. Une fois toutes les données reçues, un signal unique d'acquittement est généré. Dans ces deux cas simplifiés, il existe une hypothèse temporelle qui doit garantir que l'occurrence du signal de contrôle succède la disponibilité des données, et ceci au niveau du récepteur. On parle ici de circuits indépendants de la vitesse [14].

Si aucune simplification n'est introduite, il est possible de réaliser des circuits sans aucune hypothèse temporelle et dont le fonctionnement est correct quels que soient les retards introduits dans les fils et les opérateurs (à l'exception des fourches isochrones [7][30]). On parle de circuits insensibles aux délais ou quasi insensibles aux délais [7][14].

# 3. potentiels et coûts

## 3.1. pourquoi les circuits asynchrones sont ils attractifs ?

Nous présentons ici les potentiels généraux qu'apporte le mode de fonctionnement flot de données décrit ci-avant. On se restreint bien entendu à la réalisation de systèmes sous forme de circuits intégrés. Le paragraphe 4 montrera comment ces potentialités générales servent l'étude conjointe algorithme-architecture.

*a) Calcul en temps minimum*

Une première conséquence très importante du fonctionnement flot de données, est qu'un opérateur asynchrone peut évaluer une fonction en un temps variable (caractérisé par sa latence), compris entre une borne inférieure et une borne supérieure. Etant donné que l'opérateur implémente la signalisation de communication, la donnée peut être utilisée à sa sortie par l'opérateur qui lui est connecté, et ceci dès que possible. On imagine facilement que les variations du temps de traitement seront d'autant plus grandes que l'algorithme implémenté exhibe des dépendances fonctionnelles importantes en fonction des données. Ce point sera illustré dans la partie consacrée à l'interaction algorithmes-architectures.

En plus de cette variation qu'on peut qualifier de fonctionnelle (car uniquement liée à l'expression des dépendances présentes dans l'algorithme), il existe une autre source de variation du temps de calcul liée à l'implémentation physique. Par exemple, la convergence électrique dans un circuit intégré peut être plus ou moins rapide en fonction du nombre de transitions de zéro vers un ou de un vers zéro requises. Ceci est dû au fait qu'en CMOS les temps de traversée des transistors N et P ne sont pas identiques et donc que les transitions zéro vers un et un vers zéro n'ont pas la même durée. Ainsi, étant donné un chemin entrée-sortie emprunté par les données, le temps pourra varier en fonction des valeurs zéro ou un des données. Cette variation électrique dont il faut s'affranchir en mode synchrone n'est pas problématique en mode asynchrone.

Enfin, les caractéristiques de vitesse d'un circuit varient en fonction de paramètres qui influencent le fonctionnement des dispositifs élémentaires, tels que les variations des paramètres technologiques, la température ou la tension d'alimentation. Le fonctionnement flot de données des circuits asynchrones les rend insensibles fonctionnellement vis-à-vis de ces variations. La fin de traitement étant détectée et signalée au niveau de chaque cellule, les variations de la vitesse induites par des modifications des propriétés physiques n'altèrent pas le comportement fonctionnel.

On peut donc caractériser un opérateur par une latence minimale et une latence maximale (recherche des chaînes qui minimisent et maximisent les dépendances fonctionnelles et les temps de transfert dans les dispositifs élémentaires). Evaluer la latence moyenne est plus difficile car cela impose de modéliser les propriétés statistiques des entrées et les dépendances fonctionnelles de l'algorithme implémenté. C'est toutefois possible dans certains cas [5] [31] [39]. Mais une chose est sûre, le temps de traversée sera le temps minimum requis pour réaliser la fonction demandée, étant donné le chemin emprunté par les données (fonction de l'algorithme), la vitesse opérationnelle des dispositifs élémentaires (fonction de la technologie mise en œuvre), et les conditions de fonctionnement de la technologie (paramètres physiques dont dépend la technologie). Peut-on rêver meilleur mode de fonctionnement? Alors que la correction fonctionnelle est garantie, le circuit traite les données à la vitesse maximale.

*b) Un pipeline « élastique »*

La notion de pipeline élastique évoque un pipeline dont la capacité est variable. En asynchrone la technique du pipeline s'applique comme en synchrone, mais le nombre de données présentes dans le pipeline n'est pas imposé (*cf.* paragraphe 2.2 et 4.1). En effet, les registres de pipeline se comportent comme un tampon de type « FIFO », c'est-à-dire que les données progressent dans le pipeline aussi longtemps qu'elles ne rencontrent pas de ressource occupée, et ceci indépendamment des données qui les suivent. Rappelons qu'en synchrone, c'est l'occurrence d'un front d'horloge sur tous les registres de pipeline qui provoque le déplacement des données. L'horloge impose donc une synchronisation relative forte des données entre elles. Une fois entrées dans le pipeline, deux données sont toujours séparées par le même nombre d'étages. Ce n'est pas le cas en asynchrone et nous verrons que cette propriété ouvre des perspectives très intéressantes (*cf.* chapitre 4).

*c) Un support fiable pour les traitements non déterministes*

Traiter un signal asynchrone de l'horloge est une opération périlleuse pour un circuit synchrone. L'exemple typique est le traitement des signaux externes d'interruption dans un microprocesseur. L'échantillonnage d'un signal asynchrone peut faire apparaître un état métastable dont la durée est indéterminée, non bornée [9]. Or les circuits synchrones imposent à tous les signaux un temps d'établissement borné. Ils ne permettent donc pas d'assurer un comportement correct dans cent pour cent des cas. Généralement des mécanismes à division d'horloge sont mis en oeuvre pour limiter la probabilité d'apparition d'états métastables. En asynchrone par contre, il est possible d'attendre autant que nécessaire la fin de l'état métastable (on rappelle que la correction fonctionnelle ne dépend pas des temps de traitement). On peut donc utiliser des montages qui permettent d'assurer l'établissement d'un état stable, sans se préoccuper de la durée de l'état métastable intermédiaire. On peut par exemple sans risque d'erreur arbitrer deux signaux de requête s'adressant à une ressource unique. On peut également traiter de façon fiable l'occurrence d'un signal externe comme un signal d'interruption, pour le synchroniser avec l'exécution en cours [40].

*d) Modularité*

Les circuits et systèmes asynchrones sont véritablement modulaires. Cela est du à la localité du contrôle et à l'utilisation par tous les opérateurs d'un protocole de communication unique. Il est en effet très facile de construire une fonction complexe en connectant des modules préexistants, conçus séparément par des équipes différentes. Nous en avons fait l'expérience pour la conception du circuit Amphin qui intègre 256 processeurs élémentaires [1][10]. Si de plus, le contrôle des modules ne fait pas d'hypothèse temporelle pour garantir le fonctionnement (circuits insensibles ou quasi insensibles aux délais), alors il n'est pas nécessaire de contraindre les phases de placement et routage (si ce n'est pour des questions de vitesse).

*e) L'absence d'horloge*

Un avantage très souvent cité comme prépondérant des circuits asynchrones est la suppression des problèmes liés à la manipulation d'horloges. Cela tient au fait que les technologies modernes permettent la conception de circuits de plus en plus complexes utilisant des dispositifs de plus en plus rapides. L'enjeu se situe dans la réalisation des interconnexions. La gestion des horloges, comme la gestion de tout signal global, devient une question de tout premier plan pour exploiter les performances des technologies modernes. Les techniques et outils de conception des circuits d'horloge évoluent pour caractériser de plus en plus précisément les temps d'arrivée des horloges sur les bascules d'un circuit.

Les circuits asynchrones n'utilisent pas d'horloge globale. Les éléments de synchronisation ou contrôle sont distribués dans l'ensemble du circuit et sont ainsi beaucoup plus faciles à maîtriser. De plus, pour les circuits asynchrones dont le fonctionnement est indépendant des retards qui peuvent être introduits sur les lignes, le problème de « gigue » (clock skew) est inexistant. L'optimisation de la vitesse de fonctionnement d'un circuit asynchrone porte donc sur l'optimisation des cellules fonctionnelles elles mêmes et n'est pas contrainte par la conception d'un mécanisme d'horloge séparé. La synchronisation globale, fonctionnelle et temporelle, introduite par l'horloge est remplacée par une synchronisation locale atemporelle qui garantit uniquement le traitement séquentiel des événements.

Un autre bénéfice de la distribution du contrôle dans toute la structure du circuit est que les problèmes de pic de consommation sont écartés. En effet, l'activité électrique d'un circuit asynchrone est mieux répartie dans le temps que celle d'un circuit synchrone. Il n'existe pas d'instant prédéfini pour activer un opérateur comme c'est le cas aux fronts de l'horloge. Ici encore c'est un artefact introduit par la synchronisation globale, qu'on peut qualifier de synchronisation électrique. La consommation dans le circuit est donc bien mieux répartie dans le temps ce qui limite considérablement le bruit dans les lignes d'alimentation.

*f) Migration*

Les technologies évoluant rapidement, les industriels sont souvent confrontés à des problèmes de migration de circuits, sous-circuits ou librairies. On peut aussi définir la migration technologique d'une façon plus large en y intégrant les changements de styles de conception, mer de portes, semi-dédié, dédié.

Les circuits asynchrones se prêtent facilement à ces différentes migrations technologiques et les rendent aussi plus « souples ». Rappelons que le comportement fonctionnel d'un circuit asynchrone est indépendant de la réalisation des cellules qui le constituent pourvu que le protocole de communication soit respecté. Ainsi, au niveau le plus bas il est possible de modifier l'implémentation ou la technologie des cellules de base sans modifier la fonction. La vitesse de fonctionnement obtenue sera simplement la vitesse maximale permise par la nouvelle technologie étant donné la structure du circuit. La procédure de migration peut également être progressive. Il est possible de migrer un sous-ensemble de cellules sans que la fonction du circuit soit modifiée. On peut par exemple reconcevoir un opérateur du chemin le plus fréquent en style dédié et le substituer à l'ancien. Tant que la spécification fonctionnelle et les interfaces ne sont pas modifiées, le circuit global reste fonctionnellement correct mais pourra être plus rapide, d'une surface plus faible ou moins consommant en fonction de la modification effectuée.

Au niveau système, les mêmes bénéfices existent. On peut remplacer un circuit intégré asynchrone par son équivalent fonctionnel plus rapide sans avoir à reconcevoir le circuit imprimé. L'ensemble du système tirera partie des performances du nouveau circuit.

## 3.2. tout se paye !

*a) Une conception délicate ?*

Dans les circuits synchrones, l'échantillonnage des signaux par l'horloge permet d'ignorer tous les phénomènes transitoires qui peuvent survenir dans les parties logiques combinatoires. En asynchrone par contre toute transition d'un signal peut être interprétée comme un événement porteur d'information. Les circuits asynchrones requièrent donc une conception attentive et fine de toutes les parties du circuits (combinatoires et séquentielles) afin d'éviter toute apparition d'aléas [16]. Ce qui change fondamentalement avec l'approche synchrone est que le fonctionnement correct d'un circuit asynchrone ne doit pas dépendre de la distribution des retards dans les portes ou connexions qui composent le circuit. Pour être indépendant des temps de propagation, la progression des signaux dans les éléments logiques du circuit est contrôlée par les signaux de poignée de main. C'est le prix à payer pour supprimer la synchronisation globale réalisée par un signal d'horloge. Certaines réalisations asynchrones parviennent à relâcher la contrainte de correction fonctionnelle indépendamment des délais en introduisant des hypothèses temporelles qui conduisent à des réalisations plus simples. Il existe ainsi un éventail de techniques de conception de circuits asynchrones qui se caractérisent par leurs hypothèses temporelles respectives. La figure 3.1 présente les différents types de circuits asynchrones suivant un axe orienté vers des hypothèses temporelles de plus en plus fortes. Il faut remarquer que plus les hypothèses sont fortes plus le circuit est simple, mais moins il est robuste vis-à-vis des étapes de conception (placement, routage), et des conditions de fonctionnement (température, tension...).

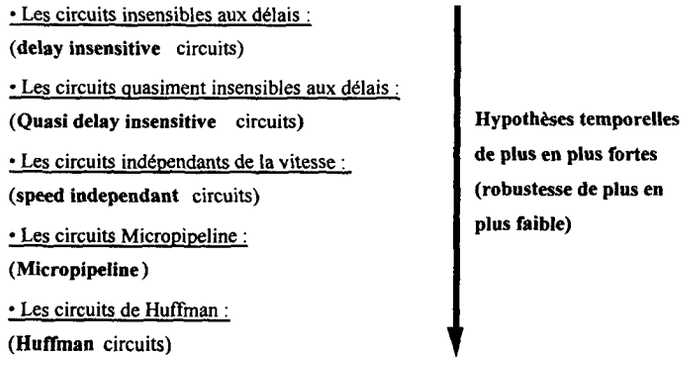



**Figure 3.1. – Terminologie des différentes classes de circuits asynchrones.**

Il existe aujourd'hui peu d'outils de conception dévolus à la synthèse de circuits asynchrones et aucun n'est commercialisé. Philips possède l'environnement de conception le plus complet [11]. D'autres industriels ont développés des outils pour répondre à des développements ponctuels dans le domaine [12] [13] [38]. Les laboratoires universitaires possèdent des outils de haut niveau mais qui souffrent d'une faible compatibilité avec les environnements commerciaux (consulter la page Web suivante pour en savoir plus sur les outils de conception universitaires : http ://www.cs.man.ac.uk/amulet/async/async_tools.html). Leurs usages restent donc privatifs ou limités à la communauté universitaire. La plupart des outils génèrent des circuits indépendants de la vitesse. Les outils de Caltech génèrent des circuits quasi insensibles aux délais [7]. Les outils de Philips permettent de générer des circuits quasi insensibles aux délais et indépendants de la vitesse [8].

D'une façon générale, le problème de la synthèse logique de circuits asynchrones est plus complexe que dans le cas synchrone. Par contre la conception des circuits complexes est facilitée pour les raisons déjà citées (modularité, absence d'horloge). Pour en savoir plus sur les méthodologies de conception de circuits asynchrones les articles ou ouvrages suivants peuvent être consultés [14] [15] [16] [17] [18].

*b) Parlons coûts*

Le style de conception asynchrone ne requiert pas de connaissance *a priori* sur les temps de propagation dans les éléments des circuits. Nous l'avons vu l'implémentation est correcte quels que soient ces temps de propagation. Les réalisations sont alors très robustes vis-à-vis de la variation des paramètres physiques. Le prix à payer est que toutes les actions doivent être acquittées. Détecter la fin des actions représente le surcoût d'une implémentation asynchrone par rapport à une implémentation synchrone. C'est un coût qui se paye au niveau de la complexité et de la vitesse.

Connaître les temps de propagation permet une implémentation efficace. En effet, si les temps de calcul de toutes les actions sont connus, il est facile d'utiliser un signal d'horloge unique qui termine et débute les actions. Le prix à payer provient du doute. Connaître les temps de propagation suppose de bien connaître les paramètres physiques, et une bonne caractérisation des dispositifs élémentaires. L'incertitude liée à la phase de caractérisation rejaillit sur la connaissance des temps de propagation et limite la robustesse des réalisations vis-à-vis des variations des paramètres physiques. En conséquence, il existe un doute, le doute que les hypothèses faites sur les temps de propagation soient fausses. Ainsi, les concepteurs de circuits synchrones contraignent la période d'horloge de façon à prendre en compte une marge d'erreur. Cette marge d'erreur peut devenir importante avec l'avancée des technologies.

Ignorer les temps de propagation permet de concevoir et utiliser des opérateurs dont le temps de calcul est variable. L'implémentation asynchrone libère le concepteur de l'approche « pire cas » des circuits synchrones, qui consiste à toujours dimensionner le circuit par rapport au temps de propagation le plus long. Il n'existe en effet qu'un seul temps de référence en synchrone, « le temps pire cas », parce qu'il n'existe qu'un signal de contrôle global : l'horloge. La distribution de ce signal global dans tout le composant peut coûter cher. Le déphasage introduit par le réseau de distribution de l'horloge doit être pris en compte dans

le dimensionnement de la période de fonctionnement du circuit. La distribution de l'horloge devient de plus en plus critique dans les circuits modernes.

En conclusion, les circuits asynchrones sont pénalisés par la nécessité de générer des signaux d'acquittement, alors que les circuits synchrones sont pénalisés par les délais introduits par la distribution de l'horloge et par la marge d'erreur introduite dans la phase de conception. Pour les circuits haute vitesse, le problème est de savoir, étant donné une technologie, quel est le style de conception qui introduit la pénalité la plus faible. Qu'en est-il pour les circuits faible performance ou faible consommation ? Faut-il choisir la simplicité de la conception synchrone ou la modularité, le calcul en temps variable, l'activité conditionnelle du style asynchrone ? Malgré les nombreux circuits asynchrones conçus, il est très difficile de trancher aujourd'hui. De notre point de vue, les circuits asynchrones constituent une alternative de réalisation qu'il convient de considérer lorsqu'on souhaite intégrer une application. Bien connaître les potentiels et coûts des deux techniques permettra d'adapter l'implémentation à la spécification, quitte même à concevoir des circuits mixtes synchrones et asynchrones.

# 4. interaction algorithmes-architectures

La communauté asynchrone est unanime : le style asynchrone offre un spectre de solutions matérielles plus étendu que le style synchrone et permet une exploration plus poussée de l'espace des solutions. L'objectif de ce chapitre est de faire partager ce sentiment au lecteur. Oublions un instant les contraintes liées à la faisabilité des circuits asynchrones (elles ont toutes des solutions adaptées) et essayons d'analyser en quoi le mode de fonctionnement « asynchrone » favorise une étude conjointe des algorithmes et des architectures.

Notre position vis-à-vis du concept d'interaction algorithme-architecture est qu'il faut tenter d'unifier les notions d'algorithme et d'architecture. Nous introduisons ici la notion de convergence algorithme-architecture. Est-il réellement nécessaire, étant donné un problème à résoudre et à implémenter, de considérer d'une part l'algorithme, c'est-à-dire une description formalisée de la spécification fonctionnelle, et d'autre part l'architecture, c'est-à-dire une seconde description de la fonction qui se différencie de la première parce qu'on sait lui faire correspondre une implémentation matérielle ? La notion d'interaction algorithme-architecture exprime le jeu d'actions qu'exerce le concepteur sur l'algorithme et son implémentation. Ce jeu d'actions est régi par des contraintes issues de la problématique d'implémentation d'une part et des contraintes d'ordre fonctionnel liées à la spécification d'autre part. C'est une interaction entre contraintes. Nous pensons que favoriser l'étude conjointe revient à favoriser l'expression conjointe de la fonction et des enjeux d'implémentation. Il nous paraît ainsi essentiel de disposer d'un formalisme vers lequel on puisse transposer la spécification fonctionnelle tout en permettant une forte lisibilité des structures matérielles sous-jacentes (parfois dénommé partitionnement). Le problème une fois formalisé, on peut appeler cela un programme, peut indifféremment être dénommé algorithme ou architecture. Pour que cette convergence algorithme-architecture soit effective pour le concepteur, il faut que le formalisme ou le langage soit d'une grande expressivité afin de favoriser la créativité algorithmique, mais il faut aussi qu'il offre une bonne lisibilité du matériel sous-jacent ou induit (c'est-à-dire de la structure matérielle). La même idée guide le programmeur en « C » qui connait les procédures de compilation et d'optimisation du code assembleur généré (la cible est alors une architecture programmable). Ainsi, la description doit à la fois rendre compte de la spécification fonctionnelle et du degré de complexité de son implémentation.

Sur la base de notre expérience dans le domaine de la conception de circuits intégrés asynchrones, nous prétendons que les langages basés sur les réseaux de processus communicants associés au mode de réalisation asynchrone favorisent cette vision de convergence des algorithmes et des architectures. Notre approche de la conception matérielle est donc focalisée sur la spécification et sa programmation dans un langage adapté (au sens qui vient d'être défini). Le langage que nous utilisons est inspiré du langage CSP [2] introduit par C.A.R Hoare pour décrire des systèmes parallèles sous forme de processus séquentiels communicants, et des commandes gardées de Dijsktra [19] [20]. Des méthodologies de conception de circuits asynchrones basées sur cette approche (VLSI programming) ont été introduites par Alain Martin, Caltech [7] et plus tard par Kees Van Berkel, Philips [8].

Les parties suivantes tentent d'illustrer cette approche de la conception de systèmes matériels et d'en démontrer les bénéfices dans le contexte AAA.

## 4.1. pipeline asynchrone

Nous essayons dans cette partie de démontrer la richesse du concept de pipeline dans le contexte des circuits asynchrones. Nous introduisons tout d'abord le pipeline asynchrone en le situant par rapport au pipeline synchrone. Nous illustrons ensuite la richesse du concept sur des exemples d'architectures complexes.

Qu'est ce qu'un registre de pipeline dans un circuit asynchrone ? Afin d'introduire rapidement le formalisme utilisé pour exprimer une spécification, considérons le processus suivant :

$$REG(A, B) = *[A?x; B!x]$$

Ce processus nommé $REG$, lit une valeur sur le canal d'entrée $A$ et place cette valeur dans la variable locale

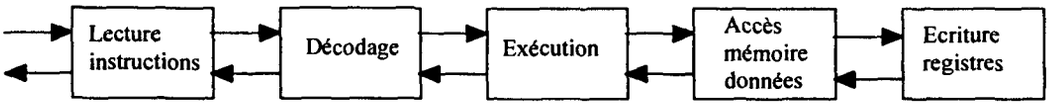



**Figure 4.1. – Un exemple de pipeline.**

« $x$ » ($A?x$). Une fois cette action terminée, c'est ce que spécifie l'opérateur de séquentialité noté « ; », le programme émet la valeur présente dans la variable « $x$ » sur le canal de sortie $B(B!x)$. La structure « $*$[suite d'actions] » spécifie que la suite d'actions est exécutée indéfiniment (boucle infinie). Ce programme réalise donc indéfiniment la copie d'une donnée lue sur le canal d'entrée $A$ vers le canal de sortie $B$. La sémantique de communication par les canaux est du type « passage de messages », ou flot de données. Si on instancie une chaîne de « $n$ » processus $REG$ on réalise une file « $FIFO$ » (« first in first out ») :

$$FIFO =$$

$$REG_0(A, B_1)//REG_1(B_1, B_2)//...//REG_n(B_n, B)$$

L'opérateur dénoté « // » est l'opérateur de concurrence. Il s'agit d'un réseau linéaire pipeliné dans lequel la logique combinatoire est réduite à la fonction identité. Le fonctionnement de ce réseau est plus riche qu'une suite de registres « Maître-esclaves » synchronisés par une horloge qui n'implémente qu'un simple registre à décalage. C'est pourtant la structure asynchrone la plus simple. Qu'est ce qui différencie les deux structures? C'est principalement la synchronisation entre les données. Dans le cas synchrone, l'horloge impose une propagation simultanée des données. C'est vrai pour cet exemple simpliste de réseau linéaire mais c'est aussi le cas dans des structures plus complexes. Dans le cas asynchrone les données se propagent indépendamment les unes des autres sans aucune forme de synchronisation cachée. La seule synchronisation qui existe est celle qui a été spécifiée à savoir la relation d'ordre entre les données. Dans le cas de réseaux pipelinés on parle de pipeline élastique (*Cf.* § 3.1.b). Ce concept illustre la liberté des informations à progresser les unes par rapport aux autres dans la structure.

D'un point de vue général on se rend compte qu'avec les systèmes asynchrones le concepteur est totalement maître de spécifier la synchronisation alors que dans le cas synchrone il doit composer avec une contrainte forte de synchronisation globale imposée par l'horloge unique. Il existe des applications pour lesquelles la synchronisation globale du mode synchrone est handicapante car elle doit être contournée lors de l'implémentation. C'est le cas par exemple des systèmes qui traitent un flot de données irrégulier ou qui possèdent un débit d'entrée différent du débit de sortie (on peut citer les systèmes de codage dont le flux d'information en sortie dépend du contenu informationnel des objets à traiter, images, sons, etc...). La quantité d'information présente dans la structure peut alors varier dynamiquement au cours du temps et le mode asynchrone ouvre des perspectives intéressantes d'algorithmes et/ou d'architectures. Le problème du dimensionnement de la capacité de mémorisation reste le même, mais le fait de relâcher les contraintes de synchronisation permet d'envisager des structures nouvelles et bien adaptées au problème à résoudre. En particulier, la charge de calcul qui est liée à la fréquence d'entrée des informations pourra être parfaitement répartie dans le temps. Il s'agit bien dans ce type de problèmes de garantir qu'en moyenne le système est capable de traiter le flux d'information entrant tout en sachant absorber les pics d'activités. Le style asynchrone convient parfaitement à l'analyse et l'implémentation de cette catégorie de problèmes.

Si on parle du problème général de l'intégration ou l'implémentation de systèmes de traitement de l'information, le style asynchrone permet de spécifier et d'implémenter de façon très élégante et efficace des algorithmes que se caractérisent par un flux d'entrée dont la densité d'information pertinente est variable et dont la charge de traitement dépend elle même de cette densité d'information. Par exemple, un système capable de segmenter des images en objets pour les coder possède ces caractéristiques. Les algorithmes de codage multirésolution font également partie de cette classe. Etant donné une consigne de débit en sortie, c'est-à-dire un budget en temps de traitement (imposée par le canal ou la qualité du récepteur – terminal –), le système peut produire une information raffinée à souhait en sortie. Dans toutes ces applications, l'environnement peut contraindre dynamiquement le système pour qu'il rende un service qui lui est adapté. Cela est possible en exploitant dynamiquement les dépendances fonctionnelles inhérentes à l'algorithme ou les dépendances vis-à-vis des données. Ce concept ouvre des perspectives de développement d'algorithmes qu'on peut qualifier de robuste vis-à-vis de contraintes élaborées dynamiquement par l'environnement [40].

Illustrons la notion de pipeline dans le cas d'une architecture complexe. Nous prenons l'exemple d'un microprocesseur. Le premier étage lit une instruction en mémoire, le deuxième la décode, le troisième l'exécute, le quatrième accède à la mémoire données et le dernier écrit dans les registres (*Cf.* Figure 4.1). Bien que cette découpe soit classique elle n'est pas fondamentale pour notre propos. Un des enjeux pour la conception des microprocesseurs haute performance est de traiter efficacement les ruptures de séquence (instructions de saut et de branchement non retardé).

Le pipeline élastique du mode de fonctionnement asynchrone permet d'envisager des solutions intéressantes. Alors qu'en synchrone toute instruction qui pénètre dans le pipeline doit être acheminée à la sortie, en mode asynchrone on peut imaginer de supprimer des instructions au cours de leur traitement. On parle d'évanouissement. Si une instruction de branchement est reconnue, les quelques instructions qui sont déjà entrées dans le pipeline (leur nombre dépend de la profondeur du pipeline)

peuvent être purement et simplement supprimées. Cette décision est prise par l'étage de décodage qui provoque l'évanouissement de l'instruction créant ainsi une bulle dans le pipeline sans perturber la progression des instructions qui précèdent. A cause de la dépendance structurelle imposée par l'horloge unique entre les instructions dans les microprocesseurs synchrones, le matériel doit gérer l'acheminement des instructions qui suivent le branchement en garantissant qu'elles n'auront pas d'effet sur l'état interne du microprocesseur (registres). Les bénéfices d'une implémentation localement synchronisée (asynchrone) apparaissent à deux niveaux : le matériel est simplifié et la consommation réduite.

Si on généralise, le pipeline élastique permet de gérer dynamiquement le taux d'occupation des ressources. Le matériel peut dynamiquement supprimer ou générer à souhait des données dans la structure. Le concept d'élasticité est directement lié à l'activité fonctionnelle de l'algorithme et donc aux dépendances fonctionnelles qu'il intègre, lesquelles peuvent évoluer dynamiquement.

On a vu les bénéfices fonctionnels liés au concept de pipeline en asynchrone, considèrons maintenant l'aspect pratique. Accroître la profondeur du pipeline n'affecte pas les performances (ceci est dû à la localité des communications). Le taux d'accélération est donc linéaire vis-à-vis du taux de pipeline sans facteur correctif lié à une pénalité d'implémentation comme c'est le cas en synchrone (diffusion de l'horloge).

En conclusion, la généralisation de la notion de pipeline élastique en asynchrone, permet l'implémentation d'algorithmes possédant des dépendances fonctionnelles complexes pouvant évoluer dynamiquement. La réalisation est d'autant plus facile qu'on peut tirer partie de la modularité des circuits ou systèmes matériels asynchrones (*Cf.* § 3.1.d).

## 4.2. distribution du contrôle

Nous analysons dans ce paragraphe les avantages structurels qu'apporte la nature locale et distribuée du contrôle des réalisations asynchrones. Nous commençons par introduire une famille d'architectures qui a été très étudiée et qui s'adresse à un grand nombre d'applications : les anneaux auto-séquencés. Nous montrerons ensuite à partir d'un exemple très simple, comment la localité du contrôle permet de générer un grand nombre de solutions algorithmiques et/ou matérielles.

*Les anneaux auto-séquencés*

Les architectures en anneaux auto-séquencés (self-timed rings) constituent une classe d'architectures qui permet à la fois d'illustrer et de valoriser la distribution du contrôle des systèmes asynchrones. On entend par distribution du contrôle l'aptitude de l'architecture à gérer à un niveau de granularité fin son contrôle interne sans avoir recours à des signaux provenants d'opérateurs externes appartenant au reste du système ou du circuit. Les seuls signaux requis sont ceux nécessaires à l'implémentation de l'interface de communication choisie et au respect de sa sémantique. Il n'y a en aucun cas besoin de signaux globaux du type horloge globale au sens des circuits synchrones.

La figure 4.2 présente la structure d'un opérateur en anneau. Après initialisation avec les opérandes de l'algorithme, l'anneau effectue le calcul en itérant sur les étages de traitement. La donnée circule ainsi dans l'anneau sans controle externe. Dans cet exemple le résultat n'est jamais exploité, mais il serait facile de compléter la structure pour qu'une condition d'arrêt provoque la sortie de la donnée. Dans le contexte de la conception de circuits et systèmes, le controle local confère aux architectures en anneau trois propriétés intéressantes. La première est que les signaux rapides sont confinés dans la structure et qu'ils ne complexifient pas la conception du reste du circuit. Le débit des communications pour alimenter l'anneau en opérandes et lire les résultats est faible par rapport à la fréquence des communications internes. Les opérateurs en anneaux sont donc vus du reste du système ou du circuit comme des cellules asynchrones banalisées, respectant un certain protocole de communication. On peut facilement imaginer la conception d'un réseau parallèle composé d'anneaux pour réaliser par exemple des calculs d'algèbre linéaire. Le réseau de processeurs présenté dans [1] est une illustration de cette propriété.

La deuxième propriété est qu'on peut concevoir des circuits très compacts en itérant le calcul sur des opérateurs de traitement communs et ceci sans rendre l'implémentation plus délicate. En multiplexant l'utilisation des ressources matérielles on obtient une structure plus compacte mais dont les performances en vitesse sont bien sûr plus modestes.

La troisième propriété caractérise les performances en vitesse d'un anneau. Nous l'avons vu, l'objectif est de replier la structure, ou de la projeter [21] afin d'effectuer plusieurs itérations en utilisant le même matériel. Que deviennent les performances? Si on se place dans le cas où la chaine critique n'est pas modifiée par la projection, y-a-t-il un coût inhérent à l'implémentation des communications locales entre étages? La réponse est apportée par la

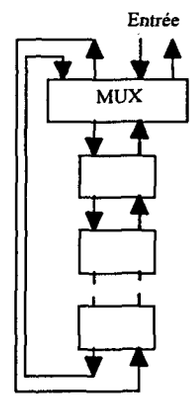



**Figure 4.2. – Structure générique d'un anneau auto-séquencé.**

théorie sur les anneaux auto-séquencés (self-timed rings) développée par Ted Williams [22] [23]. Ted Williams a démontré qu'on peut agir sur les éléments constituant l'anneau pour garantir un coût temporel nul de la gestion locale des communications dans l'anneau. Cette théorie donne donc au concepteur les moyens de définir les paramètres de l'anneau afin que son comportement soit équivalent au schéma de traitement d'une réalisation parallèle combinatoire. Ainsi, les structures en anneau permettent « d'émuler » le fonctionnement d'une structure parallèle combinatoire sans avoir recours au matériel couteûx d'une telle réalisation.

Les structures en anneaux ont été mises en œuvre pour la conception d'opérateurs arithmétiques, multiplieurs [5], diviseurs [5] [24] [25] [26]. Le concept s'applique cependant à des architectures beaucoup plus complexes, mettant en oeuvre des boucles de traitement pipelinées (multi-anneaux) [27], filtrage [28] [29], microprocesseurs.

*Une étude de cas*

On souhaite ici illustrer sur un exemple simple que la nature distribuée du contrôle favorise l'émergence d'algorithmes ou de solutions matérielles nouveaux. L'exemple est emprunté à Kees Van Berkel, Philips Eindhoven [8]. A travers cet exemple, nous souhaitons montrer que la distribution du contrôle est la source d'une grande richesse dans la formulation ou la spécification du problème initial. C'est en fait au niveau de la formulation que tout le travail de réflexion sur l'algorithme/l'architecture est mené et que des compromis entre performance, complexité et consommation sont faits par décomposition. Chaque étape de cette décomposition qui explicite la distribution du contrôle, correspond à faire un choix dans l'exploration du spectre des solutions. La réalisation étant ensuite une simple transposition de la formulation décomposée vers le matériel. La procédure de décomposition n'est pas présentée dans cet article. Elle est décrite dans [7].

La spécification du problème est la suivante : on souhaite implémenter un registre à décalage mémorisant $N$ valeurs. Les valeurs sont entrées par le port $A$ et sorties par le port $B$. L'environnement alterne les accès sur $A$ et $B$ en commençant par $B$ (*Cf.* Figure 4.3). La séquence de sortie est composée de N valeurs non spécifiées suivies de la séquence d'entrée sur A.

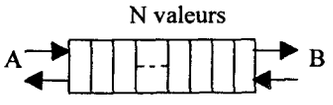



**Figure 4.3. – Vue externe du registre à décalage.**

Soit le programme suivant :

$$RD(A,B) = *[B!x; A?x]$$

Le processus $RD$ commence par écrire sur le canal $B$ la valeur lue dans la variable locale $x(B!x)$. Une fois l'action de communication sur $B$ terminée, $A?x$ spécifie une action de communication sur $A$ en lecture. La valeur lue dans le canal $A$ est affectée à la variable locale $x$. Le programme boucle indéfiniment sur cette suite d'instructions. L'opérateur $RD$ spécifie donc un étage de notre registre à décalage. Il suffit d'en instancier « $N$ » comme suit pour respecter la spécifiation initiale :

$$NRD(A,B) = RD_1(A,B_1)//RD_2(B_1,B_2)// \\ RD_3(B_2,B_3)...//RD_N(B_{N-1},B)$$

Ce programme correspond à une implémentation qui contient $N$ places mémoires équivalentes à des « Latch » (*cf.* fFigure 4.4). Le temps de cycle ou le temps qui sépare deux accès en entrée est égale à $N$ fois le temps de cycle d'un opérateur RDi. En effet, lorsque toutes les places sont occupées, le processus RDN émet une valeur en sortie, libérant ainsi une place dans le dernier processus. Cette place libre, aussi appelée « bulle », va se propager vers l'entrée en traversant les $N-1$ processus qui le précèdent. Ce n'est qu'à l'issue de cette propagation qu'une nouvelle valeur pourra être lue par $SR1$ sur le canal d'entrée $A$. Ce programme ou algorithme correspond à la réalisation la plus simple mais la plus lente. Il faut noter ici qu'une telle architecture est d'une implémentation difficile en synchrone. En effet, il faudrait construire un signal d'horloge qui circulerait en sens inverse des données de façon à émuler la propagation de la bulle.

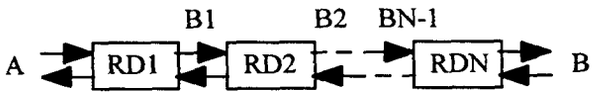



**Figure 4.4. – La réalisation la plus simple.**

Comment accélérer ce registre à décalage? Pour réduire la propagation de la bulle dans la structure on peut penser à découper cette chaîne de propagation en plusieurs parties. Cela implique d'autoriser plusieurs bulles à résider dans la structure. Soit le programme suivant :

$$RDME(A,B) = *[C!x; A?x]// *[C?y; B!y]$$

Ici le programme est composé de deux processus concurrents (*cf.* figure 4.5) contenant chacun une variable locale. Il peut être réécrit sous la forme suivante :

$$RDME(A,B) = *[[B!y//A?x]; y := x]$$

Ce processus effectue les actions de communication sur $A$ et $B$ concurremment puis transfère la valeur lue sur $A$ de la variable

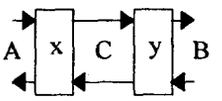



**Figure 4.5. – Structure du processus RDME.**

$x$ dans la variable $y$. Son fonctionnement s'apparente à celui d'un registre de type maître-esclave. Si on instancie $N$ processus RDME pour réaliser notre registre à décalage on obtient une structure à 2N places mémoires. Le temps de cycle est maintenant indépendant de $N$, et égal au temps de cycle d'un opérateur RDME. Ce programme correspond à la réalisation la plus couteûse mais la plus rapide. C'est l'architecture la plus facile à réaliser en synchrone (les processus RDME seraient alors des registres Maître- Esclave synchrones tous commandés par l'horloge).

Les deux algorithmes présentés ont des caractéristiques qui délimitent l'espace des solutions. On peut maintenant choisir toutes les solutions intermédiaires qui consistent à insérer des processus RDME dans une chaîne de processus RD. Dans une telle structure, il existe plusieurs chaînes de propagation des bulles qui fonctionnent en parallèle et le temps de cyle est réduit (*cf.* figure 4.6).

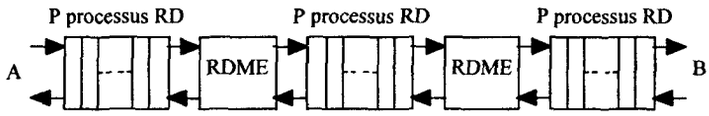



**Figure 4.6. – Une structure mixte.**

Les programmes précédents permettent de balayer un espace de solutions paramétré par le rapport complexité-vitesse. Essayons d'explorer l'espace des solutions en fonction du paramètre de consommation. Si on fait l'hypothèse que la consommation de la réalisation physique sera proportionnelle à l'activité dans le circuit, il faut chercher des algorithmes qui minimisent le nombre de transferts de données. En effet, dans tous les algorithmes proposés, toute donnée entrée traverse $N$ étages avant de se présenter à la sortie. Soit les programmes suivants :

$$SEP(A, C, E) = *[[C!x//A?y]; [E!y//A?x]]$$
$$CTR(D, F, B) = *[[B!x//D?y]; [B!y//F?x]]$$

Le processus $SEP$ est un séparateur qui lit une donnée sur le canal d'entrée $A$ et l'envoie alternativement sur $C$ puis $E$.

Le processus $CTR$ est un concentrateur qui lit les données alternativement sur $D$ et $F$ et les émet toujours sur $B$ en sortie. Soit la structure décrite Figure 4.7 qui se compose d'un séparateur, d'un concentrateur et de deux chaînes de $(N-2)/2$ processus $RD$. Dans cette structure, les données d'indices pairs transitent dans le registre à décalage du haut, alors que les données d'indices impairs transitent dans le registre à décalage du bas. Les données ne traversent donc que $(N-2)/2+2$ opérateurs ce qui diminue l'activité d'un facteur 2 environ. Le temps de cycle est également amélioré puisque les bulles ne se propagent plus que dans $N/2$ cellules.

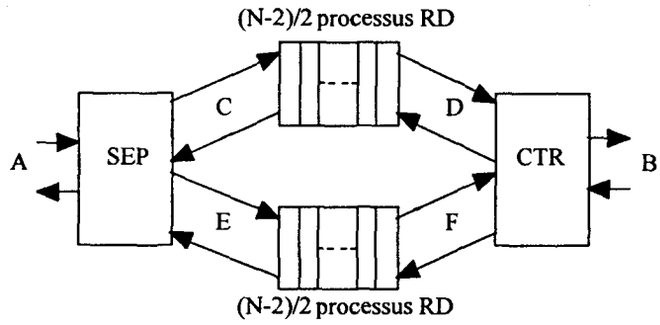



**Figure 4.7. – Une structure moins consommante.**

La technique peut être appliquée récursivement jusqu'à atteindre les performances en consommation souhaitées. On peut également imaginer d'accélérer le temps de cycle des registres à décalage en insérant des processus RDME comme proposé précédemment. Il est clair que le spectre des solutions est très grand et qu'il offre forcément au concepteur une solution adaptée à son problème.

Cette exemple montre que malgré la simplicité de la spécification, un grand nombre de solutions ou d'algorithmes peuvent être dérivés. Ces algorithmes se caractérisent par des scénarios de séquencement différents qui ont pu être imaginés grâce à la localité des communications et à la nature distribuée du contrôle.

Le formalisme utilisé nous permet d'étudier une spécification et les algorithmes qui la respectent en programmant. La lisibilité du formalisme vis-à-vis de l'implémentation matérielle n'a pas été approfondie. Nous avons donné quelques indications sur les structures associées aux programmes, mais une étude des procédures de décomposition et de compilation des programmes (c'est-à-dire la synthèse du matériel) serait nécessaire pour aller plus loin. Ce n'est pas l'objet de cet article, mais ce qu'on peut dire c'est qu'il existe une transposition directe des programmes écrits dans ce formalisme vers un matériel réalisé sous forme de circuits asynchrones. En effet, les algorithmes sont décrits sous la forme d'un ensemble de processus concurrents qui communiquent par des canaux. Ce modèle trouve un équivalent direct au niveau matériel. Il est alors facile d'imaginer la structure matérielle correspondante à un programme, c'est-à-dire l'architecture, sans avoir à la réaliser. On peut alors évaluer avec une bonne précision la complexité, les performances, voire même la consommation et effectuer des comparaisons entre algorithmes sans avoir à initier le processus de réalisation. Le point clé est que le contrôle ou le scénario de séquencement qui caractérise un algorithme est assuré par la sémantique de communication et l'existence des communications entre les processus. En manipulant le formalisme, on manipule donc conjointement l'algorithme et l'architecture. On guide alors la transformation des programmes par deux choses simultanément : le respect de la fonction et la prise en compte de contraintes d'implémentation. Ceci correspond bien à notre vision de la convergence algorithme-architecture.

## 4.3. calcul en temps moyen

Nous l'avons vu les opérateurs asynchrones effectuent leur traitement en un temps variable [5] [31] [32] [39] [40]. Cette propriété a des conséquences importantes sur l'approche de la conception d'un système asynchrone. Si on tente de généraliser, les algorithmes, quelle que soit leur complexité, peuvent exhiber des dépendances fonctionnelles internes qui dépendent des données et donc qui influencent le temps de calcul ou la consommation. On parle de dépendances fonctionnelles dynamiques. La conception des circuits asynchrones requiert donc une bonne connaissance de ces dépendances dynamiques, aspect partiellement occulté lors de la conception de circuits synchrones. Car en fait, l'existence de cette variation du temps de calcul en fonction des données ne dépend pas du mode de fonctionnement (synchrone ou asynchrone), c'est une caractéristique inhérente à l'algorithme utilisé. Ce qui change fondamentalement c'est qu'elle devient une source importante d'optimisation en mode asynchrone, et le choix de l'algorithme ou de l'architecture pour une implémentation asynchrone sera basé sur l'optimisation du temps moyen plutôt que l'optimisation du temps pire cas.

Ayant conscience de ce potentiel, le concepteur devra exprimer l'algorithme de façon à exhiber les dépendances fonctionnelles et s'assurer que la réalisation saura exploiter leurs variations dynamiques. Si on peut exploiter ces finesses des algorithmes au niveau d'un opérateur arithmétique on peut facilement imaginer les possibilités offertes par des algorithmes beaucoup plus complexes. Le réseau de processeurs décrit dans [1] et [10] démontre qu'il est même possible de changer la nature des dépendances fonctionnelles d'un algorithme pour conduire à une implémentation plus simple et plus performante.

En plus des dépendances fonctionnelles qui caractérisent les algorithmes, il est possible d'exploiter des informations sur la fréquence d'utilisation des ressources fonctionnelles d'un algorithme. Dans un algorithme complexe, il existe souvent des cas dont la fréquence d'apparition est faible. Il est alors possible de tenir compte de cela lors de la phase de réalisation. Ainsi, un chemin (fonctionnel ou matériel) rarement emprunté pourra être largement sous dimensionné sans que les performances moyennes de l'ensemble du système soient dégradées. Par contre, le concepteur pourra apporter le plus grand soin aux chemins les plus fréquemment utilisés.

Le problème du respect de contraintes temporelles peut apparaître plus complexe. Il n'en est rien. On peut toujours se ramener à une problématique pire cas et garantir le respect d'une contrainte temporelle en ne considérant que le cas critique. On peut aussi modéliser ou mesurer le temps moyen de façon à respecter une contrainte temporelle moyenne. Dans ce cas, il peut être nécessaire d'ajouter des ressources de mémorisation ou des procédures algorithmiques d'exception pour donner au système la possibilité d'absorber les « pics » d'activité (nous avons suggéré des solutions algorithmiques à ce problème dans 4.1, une solution matérielle est également proposée dans 4.4).

## 4.4. faible consommation

Asynchrone n'est pas synonyme de faible consommation. Cependant plusieurs propriétés des circuits asynchrones peuvent servir la réduction de la consommation.

– L'énergie dissipée par le système de distribution des horloges n'existe pas dans les circuits asynchrones.

– Le mode de fonctionnement asynchrone offre de façon naturelle et implicite une mise en veille à tous les niveaux de granularité. Les fonctions d'un circuit auxquelles aucune donnée n'est présentée ne consomment pas (si la technologie le permet, comme c'est le cas en CMOS). Des techniques équivalentes dans le principe ont été développées pour les circuits synchrones (arrêt des horloges dans certaines parties du circuit). Toutefois, cette économie d'activité est inhérente à la technique asynchrone, et ne requiert donc aucun outil, technique de conception ou matériel supplémentaires. Par contre, l'addition d'un système d'horloges faible consommation dans un circuit synchrone, coûte en matériel, en temps de conception et peut limiter la vitesse de fonctionnement.

– La nature distribuée du contrôle peut conduire à des gains significatifs en consommation. En effet, de nombreuses machines à états peu complexes, dont un petit nombre est impliqué dans le traitement, consomment moins d'énergie qu'un contrôleur central complexe. La logique est moins complexe (moins profonde) et les signaux sont échangés localement (réduction des capacités d'interconnexions).

La combinaison du contrôle distribué et de la mise en veille intrinsèque à ces systèmes permet d'intégrer le critère de faible consommation dès la spécification du problème (nous l'avons illustré dans l'exemple du registre à décalage). Cette approche est applicable à tout problème tel que la conception d'opérateurs arithmétiques ou de mémoires faible consommation [33]. Pour aller plus loin dans l'optimisation de la consommation on peut penser exploiter des connaissances à priori sur les signaux fournis par l'environnement au système (statistique sur la dynamique, sur la fréquence d'occurrence, etc...). Sparso propose un circuit de traitement de la parole dont l'activité du chemin de données est conditionnée par la dynamique des signaux d'entrée. Le traitement des silences s'effectue ainsi à énergie minimale [29].

– Réduire la tension d'alimentation des circuits est souvent utilisée pour limiter la puissance consommée (la puissance varie avec le carré de la tension). L'aptitude des circuits asynchrones à être fonctionnels indépendamment des temps de traversée des opérateurs élémentaires (*cf.* § 2.1), permet de réduire la tension d'alimentation avec un matériel minimum (en synchrone il faut adapter la fréquence de l'horloge à la tension). Un des exemples de robustesse le plus probant est le processeur de Caltech [34] qui peut encore fonctionner à une tension de 0.5 volt.

Il est de plus possible de faire varier dynamiquement la tension d'alimentation du circuit pour réduire la consommation. Le schéma de la figure 4.8 présente le système de traitement proposé

dans [35]. Le système évalue la charge de calcul dont il doit s'acquitter à l'aide d'un indicateur de remplissage du tampon FIFO (First In First Out) qui mémorise temporairement les données à traiter. Si la pile est presque vide le calcul est trop rapide, si la pile est presque pleine alors le calcul s'effectue trop lentement. Pour accroître ou diminuer la vitesse de traitement, le contrôle régule la tension d'alimentation de façon à ce que la partie opérative délivre la puissance de calcul nécessaire et suffisante pour s'acquitter du traitement temps réel des données. La consommation est donc auto-régulée par la quantité d'information à traiter, et le traitement est effectué à consommation minimale d'énergie. Ce dispositif illustre bien les bénéfices d'un fonctionnement flots de données implémenté à l'aide de composants robustes vis-à-vis de la tension d'alimentation.

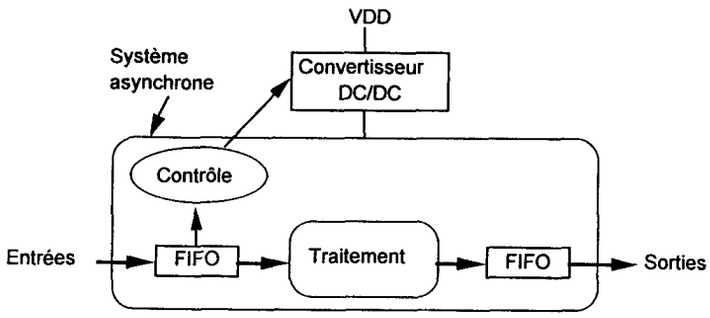



**Figure 4.8. – Régulation dynamique de la tension d'alimentation en fonction de la charge de traitement.**

Philips a appliqué cette technique à la correction d'erreurs [36]. La consommation du système est directement dépendante du taux d'erreurs présent dans le signal incident. L'algorithme de correction est tel que sa complexité est faible lorsqu'il n'y a pas d'erreur. Pour ces algorithmes, qu'on qualifie de paresseux, une implémentation asynchrone saura exploiter les irrégularités au profit de la consommation. L'approche algorithmique de ces systèmes consiste donc à favoriser les irrégularités plutôt que d'essayer de régulariser le traitement afin de l'adapter à un critère pire cas. L'approche est donc tout à fait différente de l'approche synchrone.

Pour rapprocher cette technique des idées développées dans le paragraphe 4.1, l'environnement peut ici allouer un budget consommation au système (on avait évoqué la possibilité d'allouer un budget temps). Si l'algorithme de traitement est robuste vis-à-vis de la variation de ce budget, il pourra fournir en sortie une qualité d'information en rapport avec le budget énergie consenti. On suggère ici la conception conjointe d'algorithmes et d'architectures pour les systèmes portables.

Malgré ces propriétés encourageantes, on ne peut toutefois pas conclure. Les circuits asynchrones ne sont pas dans tous les cas moins consommant que les circuits synchrones. La mise en œuvre du protocole de communication asynchrone, qui est à la base de tous les bénéfices cités, coûte plus cher que la communication à la mode synchrone (on ne peut imaginer plus simple). Pourquoi? Pour répondre il faut entrer dans les détails de l'implémentation, ce qui n'est pas l'objet de cet article. Cependant, on peut dire que la communauté asynchrone tente d'approfondir deux voies. La première est de travailler l'implémentation de façon à limiter la consommation au niveau logique. Il existe là un compromis entre la robustesse de fonctionnement et la consommation et la complexité. Les résultats ne sont pour l'instant pas probants [37]. La deuxième consiste à combiner les potentiels cités. Il faut alors étudier le système à intégrer dans son ensemble de façon à tirer partie de toutes les connaissances dont on dispose sur l'environnement, les informations à traiter et les dépendances fonctionnelles inhérentes au problème. Certaines réalisations montrent qu'un gain substantiel est possible [29] [36] [41] lorsque le système est considéré dans son ensemble.

# 5. conclusion

Dans la première partie de cet article nous avons précisé le mode de fonctionnement des circuits et systèmes asynchrones et nous l'avons situé par rapport au mode de fonctionnement des circuits synchrones. Les principes de base du mode de fonctionnement asynchrone ont été décrits pour faire le lien entre le mode de fonctionnement et la réalisation. La deuxième partie a été consacrée à une analyse générale des potentiels des circuits et systèmes asynchrones. Nous avons vu que ces potentiels existent aux niveaux fonctionnel, structurel et aussi technologique. Le prix à payer pour implémenter le mode de fonctionnement asynchrone a été abordé à la fin de la partie 3, et nous avons donné des éléments de comparaison des coûts associés aux réalisations synchrone et asynchrone.

Dans la dernière partie nous avons tenté de montrer comment le concept asynchrone pouvait servir la problématique AAA. Nous avons introduit la notion de « convergence algorithme-architecture » qui vise à aborder le problème de la conception d'un système matériel en le spécifiant à l'aide d'un formalisme unique qui explicite à la fois les contraintes fonctionnelles et matérielles. Sur des exemples simples, nous avons montré comment l'utilisation conjointe d'un formalisme adapté et du mode de fonctionnement asynchrone ou localement synchronisé conduit à une exploration poussée du spectre des solutions. Cette approche offre l'opportunité de concevoir du matériel en programmant. Une seule vue de la spécification est nécessaire : le programme, car il décrit à la fois l'algorithme et l'architecture. Nous avons illustré comment la description d'une spécification sous la forme de processus concurrents et communicants décrit la fonction tout en offrant une grande lisibilité de la structure. La transposition de ce type de programmes vers le matériel est alors directe. Il suffit au concepteur de raffiner son programme, c'est-à-dire de partitionner (« décomposer »), structurer la spécification, jusqu'à ce que la transposition du programme vers la cible matérielle

choisie permette de respecter les objectifs fixés. Cette approche permet au concepteur de conjointement élaborer l'algorithme et l'architecture.

## BIBLIOGRAPHIE


[1] F. Robin, M. Renaudin, G. Privat, N. Van Den Bossche, «Un réseau cellulaire VLSI fonctionnellement asynchrone pour le filtrage morphologique d'images», TS, ce numéro.

[2] C.A.R. Hoare, «Communicating Sequential Processes», International series in computer science, Prentice Hall, 1985.

[3] A.J. McAuley, «Four State Asynchronous Architectures», IEEE Transactions on Computers, vol. 41, n°2, February, 1992.

[4] C.D.Nielsen, «Evaluation of function blocks for Asynchronous design», EURODAC'94, Grenoble, France, pp. 454–459, Sept.1994.

[5] B. ElHassan, «Architecture VLSI Asynchrone utilisant la logique différentielle à précharge : Application aux opérateurs arithmétiques», Thèse de l'Institut National Polytechnique de Grenoble (INPG), spécialité Microélectronique, soutenue le 26 septembre 1995 à Grenoble.

[6] I.E. Sutherland, «Micropipelines», Communications ACM, vol. 32, n°6, pp. 720-738, June, 1989.

[7] A. Martin, «Synthesis of Asynchronous VLSI Circuits», Caltech-CS-TR-93-28.

[8] K. Van Berkel, «Handshake Circuits – An Asynchronous Architecture for VLSI Programming», Cambridge University Press, 1993.

[9] L. Kleeman, A. Cantoni, «Metastable behavior in digital systems», IEEE Design & Test of Computers, December 1987, pp. 4–19.

[10] F. Robin, M. Renaudin, G. Privat, N. Van Den Bossche, «A Functionally Asynchronous Array-Processor for Morphological Filtering of Greyscale Images», IEE Computers and Digital Techniques, special section on Asynchronous Architecture, vol. 143, n°5, September, 1996.

[11] K. Van Berkel, «Handshake Circuits – An Asynchronous Architecture for VLSI Programming», Cambridge University Press, 1993.

[12] A. Marshall, B. Coates, P. Siegel, «Designing an Asynchronous Communications Chip», in IEEE Design and Test of Computers, volume 11, Number 2, Summer 1994, pp. 8–21.

[13] R.F Sproull, I. Sutherland, C. Molnar, «The Counterfow Pipeline Processor Architecture», in IEEE Design and Test of Computers, volume 11, Number 3, Fall 1994, pp. 48-59.

[14] S. Hauck, «Asynchronous Design Methodologies : An Overview», Proceeding of the IEEE, vol. 83, n°1, pp. 69-93, January, 1995.

[15] «Asynchronous Digital Circuit Design», G. Birtwistle and A. Davis Editors, Springer, 1995.

[16] L. Lavagno, A. Sangiovanni-Vincentelli, «Algorithms for synthesis and testing of asynchronous circuits», Kluwer Academic Publishers, 1993.

[17] Proceedings of the second working conference on asynchronous Design Methodologies, May 30-31, 1995, London, England.

[18] M. A. Kishinevsky, A. K. Kondratyev, A. R. Taubin, V. I. Varshavsky, «Concurrent Hardware, The Theory and Practice of Self-Timed Design», Wiley Series in Parallel Computing, Chichester, 1994.

[19] E. Dijkstra, «A discipline of programming», Prentice Hall, Englewood Cliffs NJ, 1976.

[20] E. Dijkstra, «Guarded Commands, Nondeterminancy and formal derivations of programs», Communications ACM, vol. 18, August, 1975.

[21] S.Y. Kung, «VLSI Array Processors», Prentice Hall, Thomas Kailath, Series Editors, Eugleurood Cliffs, 1988.

[22] T.E. Williams, «Self timed rings and their application to division», Ph.D dissertation, Stanford university, May, 1991.

[23] T.E. Williams, «Performance of iterative computation in self timed rings», Journal of VLSI signal processing, n°7, pp. 17-31, Feb. 1994.

[24] T. E. Williams, M. Horowitz, «A zero overhead self-timed 160 ns 54 bits CMOS divider» IEEE Journal of solid state circuit, vol. 26, pp. 1651-1661, November 1991.

[25] M. Renaudin, B. ElHassan, A. Guyot, «A New Asynchronous Pipeline Scheme : Application to the Design of a Self-Timed Ring Divider», in IEEE Journal of Solid-State Circuits, vol. 31, n°7, pp. 1001-1013, July, 1996.

[26] G. Matsubara, N. Ide, «A low power zero-overhead self-timed division and square root unit combining a single-rail circuit with a dual rail dynamic circuit», in Proceedings of the third international symposium on Advanced Research in Asynchronous Circuits and Systems, April 7-10, 1997, Eindhoven, The Netherlands.

[27] J. Sparo et J. Staunstup, «Delay insensitive multi-ring structures», Integration, the VLSI journal, pp. 313–340, 1993.

[28] U.V. Cummings, A.M. Lines, A.J. Martin, «An Asynchronous Pipelined Lattice Structure Filter», in Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, Salt Lake City, Utah, pp. 126-133, November 3-5, 1994.

[29] L.S. Nielsen, J. Sparso, «A low power asynchronous data-path for FIR filter bank», in Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, March 18-21, 1996, Aizu-Wakamatsu, Japan.

[30] K. Van Berkel, «Beware the isochronic fork», Integration, the VLSI journal, n°13, pp. 103-128, 1992.

[31] A.W. Burks, H.H. Goldstine, J. von Neumann, «Preliminary discussion of the logical design of an electronic computing instrument», in Bell and Newell, Computer structures : readings and examples, Computer Science series, Mc Graw-Hill, 1971.

[32] M. Renaudin, B. ElHassan, «The Design of Fast Asynchronous Adder Structures and Their Implementation Using D.C.V.S. Logic», in Proceedings ISCAS, London, May, 1994.

[33] J.A. Tierno, A.J. Martin, «Low-Energy Asynchronous Memory Design», in Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, Salt Lake City, Utah, pp. 176-185, November 3-5, 1994.

[34] A. J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus, «The Design of an Asynchronous Microprocessor», in Charles L. Seitz editor, Advanced Research in VLSI : Proceedings of the Decennial Caltech Conference in VLSI, pp. 351-373, MIT Press, 1989.

[35] L.S. Nielsen, C. Niessen, J. Sparso, J. Van Berkel, «Low Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage», IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. 2, n°4, December, 1994.

[36] K. Van Berkel et Al, «Asynchronous Circuits for Low Power : A DCC Error Corrector», IEEE Design & Test of Computers, Summer 1994, pp. 22-32.

[37] S.B. Furber, J.D. Garside, S. Temple, J. Liu, «AMULET2e : An Asynchronous Embedded Controller», in Proceedings of the third international

symposium on Advanced Research in Asynchronous Circuits and Systems, April 7-10, 1997, Eindhoven, The Netherlands.

[38] T.A. Chu, «CLASS : a CAD system for automatic synthesis and verification of asynchronous finite state machines», in Integration, The VLSI Journal, vol. 15, n°3, October 1993, pp. 263-289.

[39] A. Tisserand, «Adéquation arithmétique architecture, problèmes et étude de cas», Thèse de l'Ecole Normale Supérieure de Lyon, spécialité Informatique, soutenue en Septembre 1997 à Lyon.

[40] M. Renaudin, «AAAA : Asynchronisme et Adéquation Algorithme Architecture», conférence invitée 3ème édition Journées Adéquation Algorithme Architecture en traitement du signal et des images, 17–19 janvier 1996, CNES Toulouse, France.

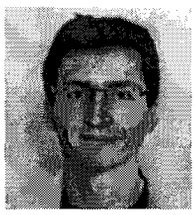
[41] K. Kessels, P. Marston, «Designing Asynchronous Standby Circuits for Low-Power Pager», in Proceedings of the third international symposium on Advanced Research in Asynchronous Circuits and Systems, April 7-10, 1997, Eindhoven, The Netherlands.

Pour avoir accès à une bibliographie complète (voire exhaustive) sur le domaine de la conception de circuits asynchrones, consulter le site Web du groupe Amulet de l'université de Manchester : http ://www.cs.man.ac.uk/amulet/

*Manuscrit reçu le 11 septembre 1997.*

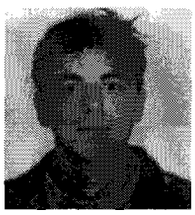
## LES AUTEURS

Marc RENAUDIN



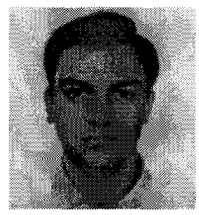
Marc Renaudin est docteur ingénieur de l'Institut National Polytechnique de Grenoble (1990). Il a rejoint l'Ecole Nationale Supérieure des Télécommunications de Bretagne, Antenne de Grenoble, en 1990, où il occupe un poste de Maître de Conférences. Il enseigne l'Architecture des Ordinateurs, les Circuits et Systèmes Asynchrones et le Traitement Numérique du Signal. En 1995, il a effectué un séjour de six mois aux Etats-Unis, dans le groupe du Professeur Alain Martin à Caltech. Ses activités de recherche portent conjointement sur le concept d'Asynchronisme et les Architectures VLSI pour l'analyse/rendu d'images. Il étudie en particulier comment l'asynchronisme peut être valorisé, de la spécification à l'implémentation, pour intégrer des systèmes complexes, performants, faible consommation.

Pascal VIVET



Pascal Vivet, né au Mans, en 1971, est diplômé de l'Ecole Nationale Supérieure des Télécommunications de Bretagne (1994). Il a débuté un doctorat de l'Institut National Polytechnique de Grenoble portant sur «L'étude et la réalisation d'une brique de base programmable asynchrone pour systèmes multiprocesseurs». Ce travail se déroule au sein du laboratoire du Centre National d'Etudes des Télécommunications de Grenoble.

Frédéric ROBIN



Frédéric Robin, né à Nantes, en 1972, est diplômé de l'Ecole Nationale Supérieure des Télécommunications de Bretagne (1994). Il a obtenu le titre de docteur de l'Ecole Nationale Supérieure des Télécommunications de Paris, spécialité «Electronique et Communications», en octobre 1997. Sa thèse a porté sur «L'étude d'architectures VLSI numériques parallèles et asynchrones pour la mise en œuvre de nouveaux algorithmes d'analyse et rendu d'images».