FPGAs virtuels : enjeux et usages

Loïc Lagadec

ComPAS'2014 : Parallélisme/ Architecture / Système
Neuchâtel, Suisse, du 23 au 25 avril 2014

FPGAs virtuels : enjeux et usages

Loïc Lagadec

Lab-STICC UMR 6532 - MOCS
ENSTA Bretagne,
2 rue Francois Verny,
29806 Brest - France
loic.lagadec@ensta-bretagne.fr

Résumé
Cet article de synthèse introduit la notion de virtualisation, appliquée aux architectures reconfigurables et en recense des cas d'usage significatifs. Ces usages balaient des préoccupations aussi diverses que la durabilité des produits (gestion de l'historique) ou le support à l'expérimentation de nouveau dispositifs (promotion de l'innovation matérielle) sans négliger la mise au point d'environnements logiciels.

Mots-clés : FPGA, virtualisation, prototypage virtuel

---

1. Introduction

La croissance remarquable des technologies reconfigurables (FPGA) depuis leur introduction par Xilinx en 1984 s'explique par leur capacité à réaliser un traitement numérique efficace (car exécuté dans le matériel) sans totalement renoncer à une flexibilité de type « logiciel ». Cette caractéristique permet d'élargir sensiblement le champ d'usage de ces technologies, et par là même d'absorber les coûts de fabrication - non récurrents mais très élevés - sur un large volume de vente.
Un point faible connu de ces technologies tient dans leur environnement de développement ; en effet les outils FPGA présentent certaines mauvaises caractéristiques, telles qu'un fort couplage à la technologie, ou encore une exigence trop élevée concernant les compétences attendues de l'utilisateur. Ainsi, l'utilisateur novice - ayant typiquement un profil d'ingénieur informatique, en poste dans une PME/PMI - est rebuté par la complexité des outils logiciels, tandis que l'utilisateur expert est, lui, potentiellement "captif", en ce sens que son expertise est intimement liée aux environnements logiciels propriétaires, mis à disposition par chaque fournisseur de solution.
Ce constat pourrait être nuancé par les récentes orientations suivies par ces fournisseurs, et le glissement progressif du domaine du tout-circuit vers l'intégration de systèmes sur puce (par exemple illustré par la famille Zinq de Xilinx[1]). Il peut en effet paraître naturel de penser que la composante logicielle des applications se renforçant (les SoC intègrent des c ?urs de type micro-processeurs), l'accessibilité de la technologie tend à s'accroitre. Cependant, la pleine exploitation des plateformes -qui motive souvent le choix de la technologie par opposition à une solution tout-logiciel- reste l'apanage d'une minorité d'usagers experts en micro-électronique.

Ces mêmes usagers subissent quant à eux une obsolescence de leurs développements antérieurs, puisque la compatibilité ascendante n'est pas assurée entre familles de FPGA au sein de l'offre d'un même fournisseur. Cela tient à l'usage de bibliothèques non portables ou dont les caractéristiques (débit, latence, ...) sont variables en fonction de la technologie ciblée. Ces mêmes usagers experts sont également les plus impactés par toute évolution ou refonte des outils constructeur, pourtant nécessaire à l'accompagnement de changements en profondeur dans l'offre matérielle.
La suite de l'article expose une réponse à cet enjeu. La section 2 tente une analyse comparée entre l'évolution de l'ingénierie logicielle et la situation des environnements d'ESL, puis introduit la notion de FPGA virtuel, dont la section 3 recense quelques usages significatifs.

2. Un éclairage logiciel sur les FPGAs virtuels

Une telle situation (fort couplage entre l'environnement de développement et le support d'exécution, faible niveau d'abstraction, durabilité hypothétique des développements, etc.) n'est pas isolée. En particulier, elle a été constatée dans le domaine logiciel dans les années 1970. La situation de l'industrie du FPGA donc doit être analysée au regard de l'évolution des pratiques dans le domaine de l'ingénierie logicielle depuis les années 1970. En particulier, les quarante dernières années ont vu l'apparition de plusieurs niveaux de machines virtuelles.

2.1. Les machines virtuelles

La généralisation des compilateurs (par exemple GCC) a permis d'offrir un outillage commun aux différentes plateformes d'exécution, à la fois facile d'accès, et offrant un usage à la carte via un jeu d'options de compilation étendu. Les architectures de type processeur, ont favorisé un schéma RISC pour faciliter l'articulation avec les compilateurs et rendre les performances plus prévisibles, tout en offrant parfois des jeux d'instructions dédiées (par exemple le MMX d'Intel) pour accélérer des traitements bien identifiés et fréquemment utilisés. Les systèmes d'exploitation, ayant la charge de l'arbitrage de l'accès aux ressources, régulent les exécutions logicielles sous contraintes de parcimonie, de performance et d'équité d'accès entre processus. L'OS isole l'application des ressources manipulée (e.g. pseudo-devices et système de fichiers) ou de leur dimensionnement (e.g. mémoire virtuelle) de façon portable (e.g. POSIX).
La large diffusion des machines virtuelles de niveau logiciel et des langages interprétés (Smalltalk, Java, ...) a permis d'élever le niveau d'abstraction du code en se détachant du modèle architectural sous-jacent du langage C. De nouveau, cet effort, en favorisant la modularité, la gestion en couches, et la prise en compte de l'hétérogénéité, a permis une portabilité et une durabilité du code applicatif accrues. L'effort de portage a été mutualisé en portant la machine virtuelle sur les différentes plateformes (OS + processeur). Ce portage étant maintenu dans le temps, la durabilité applicative est assurée.
L'évolution matérielle ayant permis de faire collaborer plusieurs processeurs ou coeurs, les environnements logiciels se sont adaptés pour en permettre une pleine exploitation. En particulier, des environnements de programmation ont été conçus pour permettre une exploitation parallèle, homogène ou non en terme d'environnement (PVM, MPI), avec prise en compte de l'interopérabilité (CORBA, IDL). Cet effort de conception s'est structuré en couches pour permettre un ré-emploi de solutions bien caractérisées (par exemple CORBA couvre les couches 5/6/7 de l'OSI et repose sur la couche réseau). Enfin ces nouveaux développements ont été transposés dans les machines virtuelles (Smalltalk DST, Java RMI, etc.). L'évolution matérielle a donc conduit à une évolution en ouverture (au sens du principe de fermeture/ouverture [18]) du logiciel avec une intégration verticale.

Enfin, plus récemment, est apparue une percée des machines virtuelles (Parallels, virtualBox, etc.) de niveau système, qui permettent de faire cohabiter plusieurs OS, l'un (l'hôte) hébergeant un ou plusieurs systèmes (dits invités) à qui on fournit l'illusion via un programme (la machine virtuelle) de s'exécuter sur une plateforme matérielle. Ces plateformes virtuelles sont aujourd'hui en phase de banalisation, et il existe par exemple des démonstrations de systèmes linux tournant dans un navigateur web [3].
Ce qui caractérise cette évolution, c'est un mouvement constant visant à se détacher du matériel, en en offrant une vue simplifiée mais significative, manipulable, et évaluable en regard de métriques, c'est-à-dire un modèle. Le niveau d'abstraction de ces modèles s'est lui même élevé au cours du temps. Un deuxième point important tient dans la nature incrémentale de ces développements, structurés en couches -potentiellement opaques- mais dont l'interface est connue, pérenne et fiable.
Cette démarche s'est fréquemment heurtée à une réticence de la part d'une partie des usagers, inquiets de la déperdition en performances brutes induite par chaque couche. Il apparait cependant que les ressources disponibles dans un système sont généralement bien supérieures au besoin réel de l'utilisateur. Ainsi ce dernier accepte volontiers d'utiliser un système invité sauf à vouloir une exploitation pleine de ses ressources (par exemple, via des applications fortement consommatrices comme les jeux vidéos).
En revanche, tant le passage à l'échelle, que la capacité de remodelage - donc la productivité dans le temps - ne se sont jamais démenties.

2.2. Le prototypage virtuel

A un niveau inférieur, le prototypage virtuel consiste à développer des systèmes matériels / logiciels sans nécessiter de prototype matériel réel, c'est à dire sans qu'un circuit imprimé, un FPGA, ou un quelconque périphérique ne soit nécessaire. Cette notion de prototype virtuel connait actuellement un regain d'intérêt avec par exemple le développement de *workshops* dédiés tels que VIPES[9].
L'avantage tient dans la possibilité d'échanger un élément pour un autre dans la configuration du système, avec des temps de mise à jour bien inférieurs à ceux atteignables avec le processus de développement traditionnel, qui requiert de re-concevoir le système dans sa globalité. De plus, la complexité croissante des systèmes embarqués, qui incluent de plus en plus de composantes cyber-physiques parallèles et distribuées, impose une phase d'exploration de l'espace de conception qui se révèlerait trop chronophage et coûteuse sans prototype virtuel.

2.2.1. Les FPGAs comme outils de prototypage

Ceci explique que, depuis quelques années, la communauté R&D cherchant à exploiter ces méthodes de prototypage a crû, en grande partie pour les industriels, sous la pression du *time-to-market*. Un des usages des FPGAs consiste ainsi à prototyper des circuits, avec à la clé, des cycles de développement raccourcis. Une fois le produit validé, il est ensuite soit réalisé sous la forme d'un ASIC si le volume de vente le justifie (coûts non récurrents élevés, coût marginal faible) ou déployé via des FPGAs, qui sont alors intégrés au système global. Cette seconde solution, présente bien sur l'intérêt de permettre par la suite une mise à jour *in the field*, c'est-à-dire sans nécessiter de campagne de retour des produits.
Pour les chercheurs en revanche, le prototypage permet en premier lieu de s'affranchir des contraintes de réalisation industrielle, et ouvre la porte à des domaines de recherche jusque là inaccessibles ; il est utilisé pour concevoir de futurs systèmes, et libère l'innovation car les hypothèses technologiques ne sont plus conditionnées par le réalisme contextuel d'un environnement industriel. Il est alors possible de concevoir en avance de phase des propositions

pouvant déboucher à terme sur des produits.

2.2.2. Les IPs reconfigurables

Dans le domaine des semi-conducteurs, une société est dite *fabless* (sans usine, sans unité de fabrication) lorsqu'elle est spécialisée dans la conception et la vente de puces électroniques, mais qu'elle en soustraite la fabrication à des sociétés spécialisées (appelées fonderies), telles que le taïwanais TSMC.
L'élévation constante des volumes de production nécessaires pour atteindre le seuil de rentabilité des équipements technologiques, explique le déséquilibre entre le nombre de sociétés fabless et le nombre de fonderies. Les sociétés fabless reposent sur un modèle économique de valorisation d'IP (Intellectual Property), c'est-à-dire le design d'une fonction électronique qu'elles protègent par des brevets ou vendent sous la forme de licences à d'autres fabricants de puces.
Parmi ces IP, on retrouve des processeurs (par exemple, ARM) et des FPGAs embarqués, disponibles pour une intégration au sein de systèmes sur puce. Ainsi, un modèle de FPGA peut être disponible sans disposer de support physique de mise en oeuvre. La mise en oeuvre intervient par raffinement de ce modèle, dit *Platform Independant Model* (PIM) en fonction de la technologie cible ou *Platform Description Model* (PDM). On obtient alors une *Platform Specific Model* (PSM), qui constitue l'IP. Cette IP requiert des outils d'exploitation (placement routage, analyse de timing, etc.) décrits en appui sur le PIM mais paramétrés par le PDM. Aujourd'hui plusieurs fournisseurs de FPGAs sont des sociétés fabless. Ainsi, la plateforme du projet FP6 Morpheus[29] était elle basée sur l'intégration de telles IPs en provenance de trois fournisseurs : M2000[23], Pact XPP[2] et DREAM[15]. Pour autant, tout changement technologique induit un coût important (à titre d'exemple de l'ordre de 3h.an pour la technologie M2000).

2.3. Les FPGA virtuels

Si en revanche, on ne souhaite pas réaliser ce portage matériel mais, à l'inverse, mettre en place une démarche réflexive (au sens de la programmation objet) d'implémentation de FPGA (invité) sur FPGA (hôte), le FPGA invité sera qualifié de FPGA virtuel ou VFPGA. Plusieurs publications adressent cette problématique depuis une dizaine d'années [12] et l'on constate un regain d'intérêt depuis 2010. [11] présente VirtualRC et propose une couche système pour le portage d'application sur différentes cibles. Zuma [5] est une proposition de vFPGA, implémentable sur Xilinx et Altera, qui repose sur le framework VTR [24].

3. Cas d'usage de la virtualisation de FPGA

Plusieurs cas d'usage émergent d'une telle démarche. En premier lieu, le portage d'architectures reconfigurables plus simples que la plateforme hôte. En second lieu, l'émulation de tout ou partie de dispositifs novateurs. A l'articulation de ces usages, on retrouve les activités de mise au point d'environnements logiciels.

3.1. Exploiter des architectures existantes

Si le portage d'architectures plus simples que la cible physique peut sembler à première vue de peu d'intérêt, il convient cependant d'en considérer deux facettes très intéressantes.

3.1.1. Faire du vieux avec du neuf : l'émulation de dispositifs obsolètes

Le premier cas à considérer est ainsi l'émulation de dispositifs obsolètes, mais présentant de la valeur (*legacy*). Cette valeur est classiquement la conséquence des efforts portés sur cette technologie pour la mise en oeuvre d'applications. On cherche alors à accroître la durée de vie des

mises en o applicatives. Cet usage est classique dans le domaine du logiciel, avec par exemple la mouvance du *retro-gaming* sous-tendue par le développement d'émulateur de plateformes commercialisées dans les années 80.
Par ailleurs, certains domaines métier, présentent des durées de vie des produits bien supérieures à ce que l'on rencontre dans le domaine grand-public. En particulier les cycles de vies constatés dans les domaines aéronautique ou de défense sont classiquement 5 à 10 fois plus longs que dans le civil. Cette longévité engendre un coût important (conservation d'exemplaires de remplacement, etc.) qui peut ainsi être supprimé à la condition de valider/certifier la conformité de la solution de remplacement.
A l'inverse, pour les produits ne nécessitant pas une reproduction à l'identique du comportement temporel, le portage sur une technologie plus avancée peut présenter des gains en performance pour peu que la différence de performances entre l'hôte et l'invité soit supérieure à la déperdition engendrée par le portage. Il est ainsi possible de disposer de *soft-cores* (processeur fourni sous forme d'IP et mis en oeuvre sur les ressources reconfigurables) plus performant que son modèle de référence.

3.1.2. Faire du complexe avec du simple : l'abstraction des ressources

De la même manière qu'une gestion de mémoire virtuelle permet aux systèmes d'exploitation de présenter des ressources plus abondantes que celles réellement présentes, en s'appuyant sur un étalement temporel de leur usage, certains travaux ont cherché à s'affranchir des limitations en ressources [7] des FPGA par une virtualisation.

3.1.3. Faire du simple avec du complexe : la conception orientée usage et/ou utilisateur

A l'inverse, un second cas d'usage tient dans la concession d'une déperdition en ressources au profit d'une facilité d'usage accrue. Comme dans le cas des OS, le constat est fait que l'abondance de ressources permet de relâcher la contrainte d'un usage parcimonieux.
Il est alors possible d'implémenter un support d'exécution adapté au besoins de l'utilisateur. Par exemple, la mise en place de matrices SIMD, bien adaptées à des traitements réguliers (traitement d'image), permet de disposer d'un modèle de calcul simplifiant la programmation de gammes d'applications. La prise en compte des architectures à plus gros grain (Coarse Grain Reconfigurable Architectures) [10] [13] s'inscrit dans la même mouvance.

3.1.4. Faire du neuf avec du vieux : l'ajout de fonctionnalités à des technologies

Un autre usage consiste à permettre une reconfiguration partielle dynamique sur des ressources ne disposant à priori pas de ce type de facilités.
Enfin, il est possible de modifier l'équilibre entre performances (surface et temps de calcul à l'exécution) et temps de calcul de la configuration. Ce glissement permet de favoriser des architectures plus gros grain ou de pré-réserver des canaux de routage pour réduire la complexité de résorption des congestions dans le réseau d ?interconnexion. Un bénéfice critique est de faciliter la synthèse à la volée (en mode *runtime*). La notion de fabrique intermédiaire [26] vise ainsi à permettre une compilation quasi-instantanée sur FPGA.
A l'extrême, on retrouve un usage généralisé des *soft-cores* pour permettre une programmation sur un mode logiciel des applications.

3.2. Le support à l'innovation

A l'opposé de ces usages, centrés sur la préservation d'un existant - matériel ou savoir faire des utilisateurs - la virtualisation permet également de prototyper des dispositifs innovants et leurs environnements logiciels.

3.2.1. Conception de dispositifs matériels

On cherche alors schématiquement à accélérer une simulation de l'invité en tirant partie de la capacité d'accélération matérielle offerte par l'hôte. Cette accélération permet de renforcer les campagnes de tests en d'en réduire la durée. Dans un contexte industriel, la pression de mise à disposition sur le marché implique souvent de renoncer à certaines phases dans le processus de développement et cet arbitrage intervient le plus souvent au détriment des tests [8] avec des coûts finaux potentiellent très élevés (un échec retentissant fut par exemple la mise en place du télescope Hubble, sans test de la lentille). Accélérer les tests revêt donc un caractère stratégique. Dans un contexte académique, les chercheurs ont besoin de mesurer l'intérêt de l'introduction d'une originalité dans la conception. Cette démonstration intervient en quantifiant un gain sur des scénarios de validation en vis à vis d'un sur-coût le plus souvent statique. A titre d'exemple, le projet ANR Ardyt [6] vise à identifier des alternatives au durcissement de la technologie, pour la réalisation de FPGAs destinés à un usage aéronautique ou spatial (donc soumis à un environnement agressif en terme de rayonnement). Pour cela l'intégration de dispositifs dédiés, destinés au diagnostic ou à la correction d'erreurs est étudiée. Cette intégration intervient en modifiant le prototype virtuel. Le sur-coût est établi lors de la synthèse du prototype étendu sur une technologie du commerce.

Un autre cas, consiste à permettre la mise en place de bitstream relogeable sur une architecture quelconque, c'est-à-dire, sans qu'elle ait été conçue dans cette optique comme le sont par exemple les eFPGAs de la plateforme FlexTile [14]. Un tel bitstream doit être perçu comme l'est une librairie .so par exemple, c'est à dire comme une portion d'application déplaçable dans l'espace d'exécution (mémoire, architecture) avec une capacité de ré-édition de liens.

3.2.2. Conception d'environnements logiciels

De la nature de l'architecture dépendent les besoins en terme d'outils. Or il apparait que les concepteurs d'architectures reconfigurables capitalisent naturellement sur des patrons architecturaux, ce qui laisse peu de place à des ruptures dans les outils. Le développement d'outils se focalise donc le plus souvent sur des couches supérieures du flot (par exemple la HLS [17]) ou sur le contrôle de l'exécution (par exemple sur un mode système [19]).

A un niveau bas, l'information technologique est difficilement accessible sinon absente. Concevoir des outils est donc cantonnée à une activité de niche.

Dans [21], Paulson et al. ont démontré un usage dynamique de spartan3, mais cette maitrise a été acquise au prix d'une rétro-ingénierie fastidieuse, destinée à isoler dans le binaire de configuration (à l'époque non crypté) l'ensemble des bits et leur signification, ressource par ressource. La prise en considération de FPGAs virtuels permet donc de favoriser la portabilité des outils [11] d'une part, mais également d'offrir un socle pérenne pour le développement d'outils innovants [5].

A bas niveau, les outils nécessaires à la programmation et l'exploitation des FPGAs (floorplanning, placement-routage, partitioner/packer, etc.) sont relativement peu nombreux [4] [27] [25] et privilégient des modèles d'architectures simplifiés.

La génération du prototype de VFPGA (incluant donc sa structuration de mémoire de configuration) permet de poser des hypothèses simplifiant considérablement la tâche aux outils d'exploitation. Une approche descendante - de type ingénierie dirigée par les modèles - permet une approche modulaire, en appui sur les raffinements (transformation de modèle à modèle) successifs. Chaque niveau peut alors constituer une pivot d'échange permettant de sortir du flot vers des outils tiers en aval ou, à l'inverse, de consommer la production d'outils tiers en amont. Émerge une infrastructure ouverte, permettant d ?aborder les étapes chronophages, et agissant en qualité de pierre angulaire pour une communauté de type open source.

3.3. Limitations

Tout ajout de couche de virtualisation induit une déperdition en performance, à la fois en terme de fréquence et de surface.

3.3.1. Prise en compte des comportement temporel

Dans un cadre de conception de type HPC, le comportement temporel attendu de l'application est généralement exprimé par le concepteur, mais des modifications itératives peuvent apparaitre pour garantir de façon implicite que ces contraintes sont remplies. La virtualisation altère le comportement temporel de l'exécution, l'effort de rafinement manuel est perdu, voire même peut dégrader les performances.

En particulier, un des avantages de la virtualisation peut tenir dans l'assurance de disposer de bitstream relogeable. Il est cependant pratiquement impossible de garantir la constance des paramètres "technologiques virtuels" (en premier lieu les caractéristiques électrique et temporelles des ressources de la couche virtuelle). Le portage d'application et la relocation de bitstream sont donc possibles mais une prise en compte de contraintes temporelles nécessite de reconstituer le jeu de paramètres technologique et de re-synthétiser les applications en tenant compte de ces paramètres.

Cette limitation n'est cependant pas pénalisante dans un cadre de synthèse de haut niveau, dans lequel le concepteur exprime des spécifications purement logicielles et donc dépourvues de notion temporelle. Le concept de FPGA virtuel ne convient donc pas à des usagers experts souhaitant tirer des performances optimales de la ressource, quitte pour cela à optimiser finement, et en grande partie manuellement, la spécification applicative. En revanche, elle présente de nombreux avantages pour un usager soucieux d'obtenir rapidement et facilement une réalisation matérielle. A l'image des VMs logicielles à ramasse miette qui ne sont pas adaptées à la programmation d'applications temps réel, l'introduction d'une couche virtuelle s'accommode bien de l'usage de synthèse de haut niveau, moins bien de la recherche de performances de pointe.

3.4. Prise en compte de la déperdition en surface

La déperdition en surface est importante (de 100x [16] à 40x [5]) mais peut être minorée par deux effets. Le premier est directement corrélé à la loi de Moore, et à la croissante exponentielle des ressources disponibles. Cette abondance a induit un glissement des enjeux, de l'usage parcimonieux, à un plein usage des ressources. La déperdition est donc moins problématique, d'autant que la facilité d'usage offerte par la virtualisation maximise l'usage de la couche virtuelle. Par analogie, si les machines virtuelles logicielles consomment une partie des ressources, les politiques de gestion de mémoire à base de ramasse miette permettent de réduire les fuites mémoires, et le déploiement d'applications distribuées sur plusieurs coeurs est simplifié, ce qui permet de gagner en performances. Dans le contexte VFPGA, la couche virtuelle est consommatrice mais le taux de remplissage de cette couche est plus important que celui d'une architecture généraliste.

Le second effet tient dans la possibilité de spécialiser la couche virtuelle par domaine applicatif. Ainsi, certaines briques applicatives sont directement implémentées et proposées comme des primitives de l'architecture. Cette démarche renforce l'approche classique consistant à intégrer des blocs dédiés (mémoire, DSP, microprocesseur, etc.) dans les architectures. Dans le cas d'une couche virtuelle, ces ressources sont optimisées puisque implémentées sur l'hôte et ne contribuent donc pas à un sur-coût (modulo une fragmentation interne si elles ne sont pas utilisées).

3.5. Prise en compte de l'effort logiciel

Le troisième volet des coûts de la couche virtuelle tient dans la nécessité de développer des outils d'exploitation adaptés. Cependant, de tels travaux existent [28, 4] qui couvrent également la production du prototype [22, 20]. La flexibilité de ces outils permet de limiter l'effort de développement additionnel.

4. Conclusion

Il apparait donc que la notion de FPGA virtuel trouve son utilité à la fois en amont de la réalisation de dispositifs, lors d'une phase à vocation innovante ou pré-industrielle, et en aval de la rupture de disponibilité. Ces deux situations sont illustrées par la figure 1. Dans ces deux cas, l'intérêt espéré est lié au matériel (performances, disponibilité) mais ce prototype permet également de tester des innovations en terme d'outillage logiciel, et ce point est critique puisque les outils sont reconnus pour être difficiles d'accès.
Un autre effet, induit par capacité d'une définition consensuelle des attendus d'un FPGA virtuel, est de permettre de fédérer des efforts autour d'une même plateforme, ouverte et documentée. L'assurance d'une certaine pérennité est également à même d'agglomérer des usagers experts, las de subir les aléas technologiques et de constater la volatilité des développements menés sur les FPGAs du commerce. Ce constat a notamment motivé une démarche structurante, initiée dans le cadre du GDR Soc-Sip sous la forme d'une action "plateforme FPGA Virtuel".

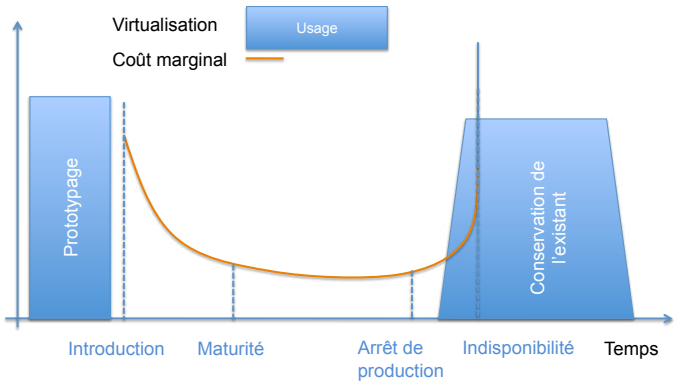

FIGURE 1 – Cycle de vie des produits et usages de la virtualisation

L'action "Plateforme FPGA Virtuel" ambitionne de structurer la communauté nationale de façon verticale (du domaine applicatif à la conception de la brique technologique) à travers une action. Les principaux attendus de cette action sont de permettre une capitalisation des travaux menés par les équipes nationales, mais aussi une évaluation chiffrée des contributions pour en faciliter le transfert technologie, vecteur de valeur ajoutée. En particulier, l'accent est mis sur la composition modulaire des contributions pour une capacité à adresser des niches, à maintenir une chaine performante facilement extensible, et à garantir de disposer de démonstrateurs et tutoriels à destination des usagers, des industriels, mais aussi des étudiants de troisième cycle. Dans cette optique, la virtualisation ne se limite pas à un effort d'abstraction de technologies commerciales (circuits Xilinx, ...) ayant pour seul objectif de s'affranchir des environnements de

programmation existants. L'objectif est d'assurer : à la fois la portabilité des développements applicatifs, mais aussi de faciliter une identification des concepts architecturaux inhérents aux FPGAs, et ainsi une réflexion sur les technologies de demain, leur usage, et leur besoin en terme d'outillage.
La couche virtuelle est garante d'une maitrise technologique totale, avec à la clé une capacité de prospection multi-dimensionnelle (architecture, structure de la configuration, usage dynamique, mise en place d'outils et de métriques, mise à disposition via le web, etc.).

Remerciements

Ce projet a partiellement été financé par le projet ARDyT (ANR-11-INSE-15).

Bibliographie

1. Zynq-7000 all programmable soc, http ://www.xilinx.com/products/silicon-devices/soc/zynq-7000/.
2. Baumgarte (V.), Ehlers (G.), May (F.), Nückel (A.), Vorbach (M.) et Weinhardt (M.). – PACT XPP : A self-reconfigurable data processing architecture. *J. Supercomput.*, vol. 26, n2, septembre 2003, pp. 167–184.
3. Bellard (F.). – Un système linux dans un navigateur web. – http ://bellard.org/jslinux.
4. Betz (V.) et Rose (J.). – VPR : A new packing, placement and routing tool for FPGA research. – In *Field Programmable Logic and Application*, *LNCS*, LNCS, 1997.
5. Brant (A.) et Lemieux (G.). – Zuma : An open FPGA overlay architecture. – In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2012.
6. Consortium (A.). – The ARDyT project : http ://ardyt.irisa.fr.
7. Fornaciari (W.) et Piuri (V.). – Virtual FPGAs : Some steps behind the physical barriers. – In Rolim (F.) (édité par), *Parallel and Distributed Processing*volume 1388. Springer Berlin Heidelberg, 1998.
8. Gallagher (T.). – Keynote from Tim GALLAGHER, lockheed martin. – In *Reconfig*, 2013.
9. Goehringer (D.) et Huebner (M.). – Workshop on virtual prototyping of parallel and embedded systems - vipes. – 2013-2014.
10. Heyse (K.), Davidson (T.), Vansteenkiste (E.), Bruneel (K.) et Stroobandt (D.). – Efficient implementation of virtual coarse grained reconfigurable arrays on FPGAs. – In *23rd International Conference on Field Programmable Logic and Applications (FPL'13)*, 2013.
11. Kirchgessner (R.), Stitt (G.), George (A.) et Lam (H.). – Virtualrc : a virtual FPGA platform for applications and tools portability. – In *FPGA*, 2012.
12. Lagadec (L.), Lavenier (D.), Fabiani (E.) et Pottier (B.). – Placing, routing, and editing virtual FPGAs. – In *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, *FPL '01*, FPL '01, pp. 357–366, London, UK, UK, 2001. Springer-Verlag.
13. Lagadec (L.), Picard (D.), Corre (Y.) et Lucas (P.-Y.). – Experiment centric teaching for reconfigurable processors. *Int. Journal of Reconfigurable Computing*, no952560, 2011, p. 14.
14. Lemonnier (F.), Millet (P.), Almeida (G.), Hubner (M.), Becker (J.), Pillement (S.), Sentieys (O.), Koedam (M.), Sinha (S.), Goossens (K.), Piguet (C.), Morgan (M.-N.) et Lemaire (R.). – Towards future adaptive multiprocessor systems-on-chip : An innovative approach for flexible architectures. – In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pp. 228–235, July 2012.
15. Lodi (A.), Toma (M.) et Campi (F.). – A pipelined configurable gate array for embedded processors. – In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on*

Field Programmable Gate Arrays, *FPGA '03*, FPGA '03, pp. 21–30, New York, NY, USA, 2003. ACM.

16. Lysecky (R.), Miller (K.), Vahid (F.) et Vissers (K.). – Firm-core virtual FPGA for just-in-time FPGA compilation. – *FPGA '05*, FPGA '05, pp. 271–271, New York, NY, USA, 2005. ACM.
17. Martin (E.), Sentieys (O.), Dubois (H.) et Philippe (J.-L.). – Gaut : An architectural synthesis tool for dedicated signal processors. – In *IEEE European Design Automation Conference*, 1993.
18. Meyer (B.). – *Object Oriented Software Construction*. – Prentice Hall, 1988.
19. Muller (F.), Rhun (J. L.), Lemonnier (F.), Miramond (B.) et Devaux. (L.). – A flexible operating system for dynamic applications. *XCell journal*, vol. 73, 2010.
20. Padalia (K.), Fung (R.), Bourgeault (M.), Egier (A.) et Rose (J.). – Automatic transistor and physical design of FPGA tiles from an architectural specification. – In *FPGA*, pp. 164–172, 2003.
21. Paulsson (K.), Huebner (M.), Auer (G.), Dreschmann (M.), Chen (L.) et Becker (J.). – Implementation of a virtual internal configuration access port (JCAP) for enabling partial self-reconfiguration on xilinx Spartan III FPGAs. – In *Field Programmable Logic and Applications*, pp. 351–356, 2007.
22. Picard (D.) et Lagadec (L.). – Fast prototyping environment for embedded reconfigurable units. – In *ReCoSoC*, pp. 1–8. IEEE, 2011.
23. Pulini (G.) et Hulance (D.). – *Dynamic System Reconfiguration in Heterogeneous Platforms : The MORPHEUS Approach*, chap. 4. – Sringer, 2009.
24. Rose (J.), Luu (J.), Yu (C. W.), Densmore (O.), Goeders (J.), Somerville (A.), Kent (K. B.), Jamieson (P.) et Anderson (J.). – The VTR project : Architecture and CAD for FPGAs from verilog to routing. – In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, *FPGA '12*, FPGA '12, pp. 77–86, New York, NY, USA, 2012. ACM.
25. Sidiropoulos (H.), Siozos (K.), Figuli (P.), Soudris (D.), Huebner (M.) et Becker (J.). – JITPR : A framework for supporting fast application's implementation onto FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2013.
26. Stitt (G.) et Coole (J.). – Intermediate fabrics : Virtual architectures for near-instant FPGA compilation. *Embedded Systems Letters*, vol. 3, 2011.
27. Teodorov (C.) et Lagadec (L.). – Model-driven physical-design automation for FPGAs : fast prototyping and legacy reuse. *Software : Practice and Experience*, 2013.
28. Teodorov (C.), Picard (D.) et Lagadec (L.). – FPGA physical-design automation using model-driven engineering. – In *Recosoc 2011*, 2011.
29. Voros (N. S.), Rosti (A.) et Hübner (M.) (édité par). – *Dynamic System Reconfiguration in Heterogeneous Platforms : The MORPHEUS Approach*. – Springer, 2009.