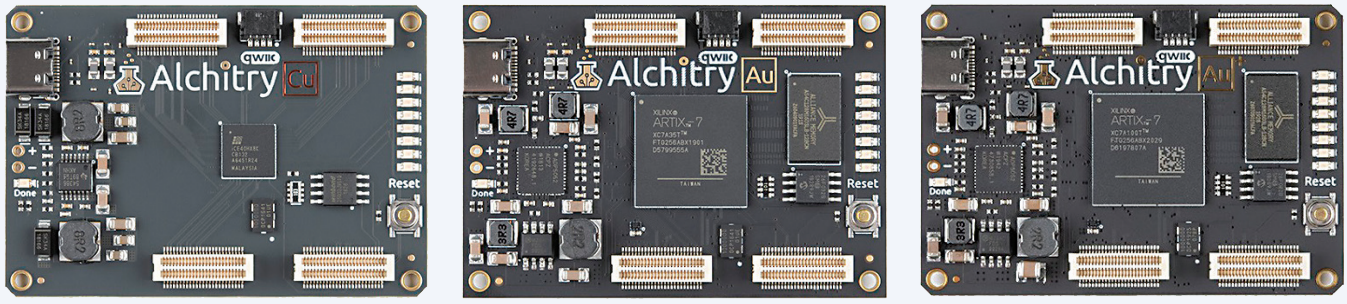
dans l'intimité d'un processeur *open source*

Extrait : résultats des FPGA Lattice et Xilinx



Cartes de développement FPGA d'Alchitry citées dans cet article. De gauche à droite : Cu - Au - Au+.

Monte Dalrymple (États-Unis)

Cet article est un extrait de l'ouvrage de *Monte Dalrymple*, « Inside an Open-Source Processor », publié en anglais par Elektor. Les électroniciens ont adopté la technique FPGA combinée au langage (V)HDL. De plus, avec l'architecture RISC-V, l'*open source* accède aux applications professionnelles. Cet article expose des comparaisons point par point et des résultats obtenus à partir d'un exemple d'application exécuté sur les cartes FPGA *Lattice* et *Xilinx* de la série *Alchitry* maintenant disponible chez Elektor. Mais d'abord, quelques conseils.

Note de l'éditeur : cet article est un extrait du livre *Inside an Open-Source Processor — an Introduction to RISC-V* formaté et légèrement modifié pour correspondre aux normes éditoriales et à la mise en page du magazine *Elektor*. Puisque cet article est extrait d'une publication plus vaste, certains termes peuvent faire référence à des passages du livre d'origine situés ailleurs. L'auteur et l'éditeur ont fait de leur mieux pour l'éviter et seront heureux de répondre aux questions – Pour les contacter, voir l'encadré « **Des questions, des commentaires ?** ».

Jusque-là, mettre en œuvre un projet enchaînant des outils FPGA était complexe, mais avec la maturité de la technique, c'est devenu bien plus simple. Pour l'exemple présenté ici (une horloge sur 24 h décrite ailleurs dans le livre), la majorité du travail est déjà accomplie.

Il faut quand même savoir que les gros projets, ou les projets exploitant au mieux le potentiel FPGA, seront à coup sûr bien plus complexes.

Dans le cas de Lattice et Xilinx, outre la sélection de la cible FPGA appropriée pour le projet, seuls deux ou trois fichiers devront être liés dans les outils. Pour cette approche simplifiée, tous les fichiers du projet doivent être dans le même dossier. Comme indiqué

ci-dessus, les lecteurs sont fortement encouragés à consulter les tutoriels Alchitry pour l'installation du logiciel FPGA et la configuration du projet car seul l'essentiel sera abordé ici.

Conseils Lattice (Alchitry Cu)

Le FPGA utilisé sur cette carte nécessite le logiciel Lattice *iCEcube2*. Il s'agit d'un produit mature sans fioritures. Voici la liste des points à retenir :

1. S'assurer de configurer le projet avec les options correctes. L'Alchitry Cu utilise le FPGA *HX8K* de la famille *iCE40*, en boîtier CB132. Tous les bancs d'E/S s'alimentent en 3,3 V.
2. Ne saisir que le fichier *yrv_alchitry.v* comme fichier de conception : tous les autres fichiers seront automatiquement inclus dans le bon ordre.
3. Pour l'outil de synthèse, veiller à sélectionner Lattice LSE Synthesis. Aucun fichier de contraintes n'est requis pour la synthèse logique.
4. Une fois la synthèse terminée, les fichiers de contrainte de broches *yrv_alchitry.pcf* et de synchronisation *timing.sdc* doivent être liés (*linked*) avant d'exécuter le reste des outils.

Des avertissements en nombre apparaîtront dans les journaux, mais aucun d'eux ne devrait nécessiter d'action.

Conseils Xilinx (Alchitry Au et Au+)

Les FPGA utilisés sur ces cartes nécessitent le logiciel *Vivado* de Xilinx. Il s'agit d'un produit complexe, et ce projet n'utilisera qu'un petit sous-ensemble des capacités de ce logiciel. Voici la liste des points à retenir :

1. Comme fichier de conception, ne saisir que le fichier *yrv_alchitry.v*.
2. Comme fichier de contraintes ne saisir que le fichier *yrv_alchitry.xdc*. Ce fichier contient les contraintes de broches, de timing ainsi que les spécifications de tension de programmation.
3. L'option de FPGA correcte est xc7a35tftg256-1 pour la carte Au et xc7a100tftg256-1 pour la carte Au+.

Des avertissements en nombre encore plus grand apparaîtront dans les journaux, mais là aussi aucun ne devrait nécessiter d'action.

Résultats FPGA

Ce projet-exemple a été mis en œuvre sur les trois cartes-cibles de développement FPGA. Les résultats individuels sont analysés ci-dessous. Il est intéressant de commencer par une comparaison point par point de certains des résultats. Le **tableau 1** montre la vitesse d'horloge rapportée pour le diviseur de 100 MHz.
Le diviseur de 100 MHz a été conçu pour une vitesse maximale avec un seul niveau de logique entre les flip-flops. Cela signifie que ce résultat devrait représenter les capacités de la technique.

Si la vitesse annoncée pour la puce de Lattice est vraie, c'est impressionnant. Ce qui est le plus intéressant, c'est que la vitesse

Tableau 1. Résultats FPGA.

Diviseur de 100 MHz	Alchrity Cu	Alchrity Au	Alchrity Au+
Vitesse (à la synthèse)	313,9 MHz	114,1 MHz	114,2 MHz
Vitesse (au placement)	542,3 MHz	–	–
Vitesse (finale)	625,4 MHz	111,9 MHz	111,9 MHz

Tableau 2. Résultats FPGA.

Attribut FPGA	Alchrity Cu	Alchrity Au	Alchrity Au+
Éléments Logiques	7 680	20 800	63 400
É.L. utilisés	4 646	2 189	2 185
É.L. utilisés (%)	60,6%	10,5%	3,5 %
Flip-flop	1 331	1 377	1 377
Vitesse (synthèse)	11,8 MHz	28,1 MHz	27,7 MHz
Vitesse (placement)	25,8 MHz	-	-
Vitesse (finale)	30,8 MHz	34,4 MHz	34,2 MHz

indiquée s'améliore et devient sans doute plus précise, en passant de la synthèse logique au placement logique puis au résultat final du routage. Même s'il vaut mieux que les performances finales soient meilleures que l'estimation initiale, un écart d'un facteur deux n'est pas idéal, car il pourrait motiver l'abandon d'un projet jugé à tort irréalisable.

Les outils Xilinx ne donnent pas d'estimation distincte de la vitesse au niveau du placement logique, mais le chiffre de la synthèse logique est quasi identique à celui du résultat final. Du point de vue d'un concepteur, c'est préférable.

Le **tableau 2** indique les ressources requises pour le projet ainsi que la fréquence d'horloge max. pour le CPU lui-même. Ces résultats ne tiennent pas compte des compteurs à 64 bits inclus dans l'*architecture RISC-V Privileged*.
La 2e ligne du tableau montre pourquoi la comparaison des résultats des FPGA peut être compliquée. D'après le tableau, l'implémentation de Xilinx nécessite moins de la moitié des ressources requises par l'implémentation de Lattice. Mais un élément logique Xilinx

Listage 1. Extrait de rapport de Lattice.

```
Total Logic Cells: 4646/7680
       Combinational Logic Cells: 3315    out of   7680    43.1641%
       Sequential Logic Cells:    1331    out of   7680    17.3307%
       Logic Tiles:               847     out of   960     88.2292%
   Registers:
       Logic Registers:           1331    out of   7680    17.3307%
       IO Registers:              0       out of   1280    0
   Block RAMs:                    12      out of   32      37.5%
   Warm Boots:                    0       out of   1       0%
   Pins:
       Input Pins:                36      out of   95      37.8947%
       Output Pins:               50      out of   95      52.6316%
       InOut Pins:                0       out of   95      0%
   Global Buffers:                2       out of   8       25%
   PLLs:                          0       out of   2       0%
```

contient beaucoup plus de logique qu'un élément logique Lattice. En outre, la conception Xilinx utilise des blocs DSP dédiés que la puce Lattice ne possède pas.

Le pourcentage d'éléments logiques utilisés révèle la disparité de taille entre les trois FPGA utilisés ici. Un peu plus de la moitié de la puce Lattice est nécessaire pour la conception *yrv_mcu*, mais environ un 10e de celle de Xilinx de milieu de gamme est utilisé et moins d'un 20e du grand Xilinx est nécessaire. Cette utilisation est conforme au coût des différentes cartes de développement.

Comme prévu, le nombre de flip-flops requis pour la conception est similaire sur les trois FPGA, et le nombre exact est plus une fonction de l'outil de synthèse logique qu'autre chose. Les outils de synthèse logique répliquent les flip-flops pour augmenter les performances ou simplifier le routage.

Les vitesses montrent que les trois FPGA fournissent à peu près les mêmes performances, c'est un peu surprenant car le FPGA de Lattice ne contient pas les blocs DSP spécialisés utilisés dans l'implémentation de Xilinx.

Alchitry Cu en détail

Extrait directement du rapport de mise en œuvre de Lattice, le **listage 1** montre les détails d'utilisation des ressources. Extraite d'une capture d'écran de l'outil *Lattice floorplanner*, la **figure 1** montre l'utilisation des éléments logiques tels que répartis sur le FPGA. Cette distribution est assez uniforme sur l'ensemble, même si l'utilisation globale n'est que de 60 % environ.

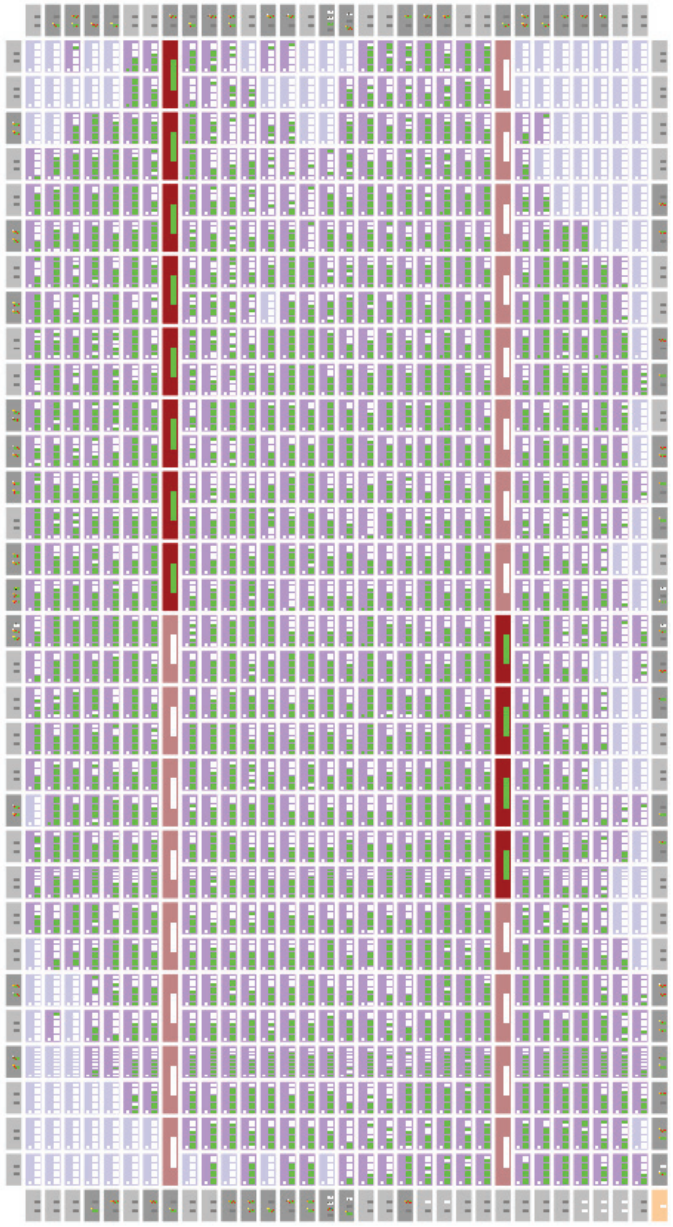


Figure 1. Vue du Floorplanner de Lattice (Alchitry Cu).

Alchitry Au en détail

Extrait directement du rapport de mise en œuvre de Xilinx, le **listage 2** montre les détails d'utilisation des ressources. Le rapport complet de Xilinx est beaucoup plus volumineux et couvre tout le matériel dédié disponible dans le FPGA.

La **figure 2** vient d'une capture d'écran de l'outil Xilinx floorplanner qui montre l'utilisation des éléments logiques répartis sur le FPGA, ainsi que la manière dont les blocs logiques dédiés y sont

Listage 2. Extrait de rapport de Xilinx XCA35T.

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              | 2189 |     0 |     20800 | 10.52 |
|   LUT as Logic          | 2189 |     0 |     20800 | 10.52 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         | 1377 |     0 |     41600 |  3.31 |
|   Register as Flip Flop | 1377 |     0 |     41600 |  3.31 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    3 |     0 |     16300 |  0.02 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

répartis. Il semble que les outils de placement débutent toujours par le coin inférieur gauche d'un FPGA. La logique semble divisée en deux parties, mais la raison n'est pas claire.

Alchitry Au+ en détail

Aussi extrait directement du rapport de mise en œuvre de Xilinx, le **listage 3** montre les détails d'utilisation des ressources. On note que les chiffres relatifs aux ressources utilisées sont presque identiques entre les deux FPGA Xilinx.

La **figure 3** vient aussi d'une capture d'écran de l'outil Floorplanner de Xilinx. Il est ici intéressant de noter que même si ce FPGA contient trois fois plus d'éléments logiques, la surface globale requise par l'exemple choisi semble être répartie sur une surface quasi identique à celle de la 1ère implémentation Xilinx.

Programmation matérielle

Avec le chargeur Alchitry, le téléchargement du *bitstream* (suite de bits) FPGA dans une carte de développement Alchitry est simple. Ce chargeur autonome est automatiquement installé avec le logiciel *Alchitry Labs* disponible auprès d'Alchitry. La **figure 4** montre l'interface utilisateur du chargeur.

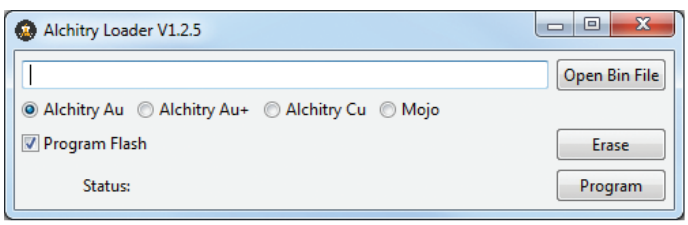


Figure 4. Vue du chargeur d'Alchitry.

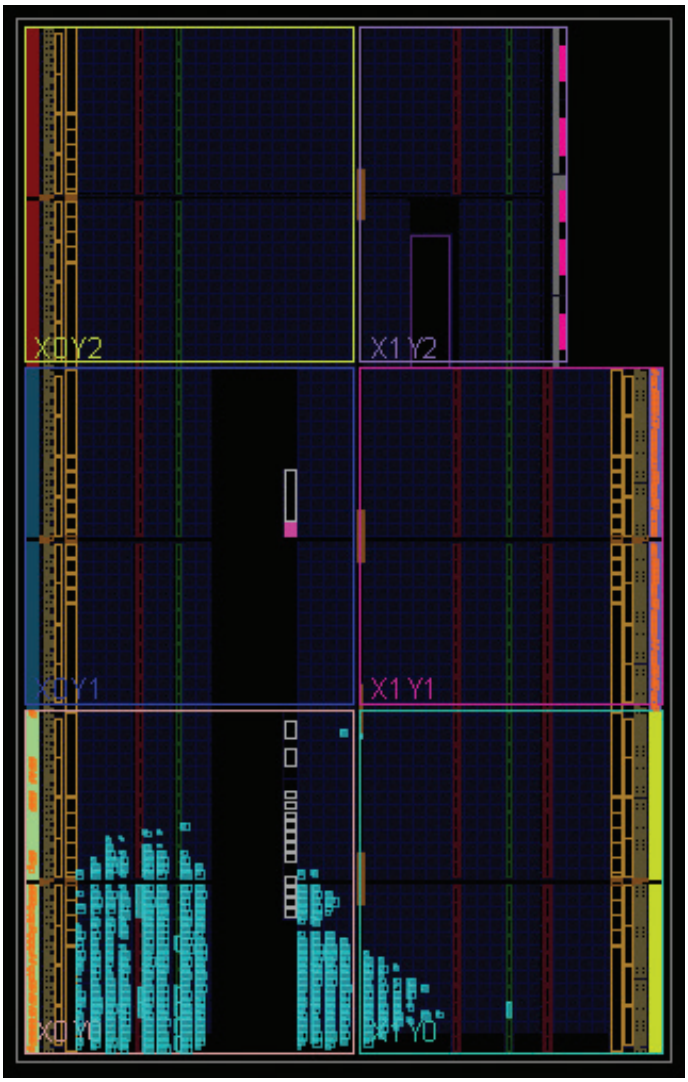


Figure 2. Vue du Floorplanner de Xilinx XCA35T (Alchitry Au).

La programmation de la mémoire Flash de la carte de développement utilisée pour charger le FPGA se résume à spécifier le fichier bitstream, sélectionner la carte cible et cliquer sur le bouton *Program*.

Par défaut, le programme recherche un fichier bitstream de type *.bin*, utilisé par défaut par le logiciel iCEcube2. Le logiciel Vivado produit un fichier bitstream de type *.bit*, il faut en tenir compte pour charger le fichier bitstream de Xilinx.

210347-04

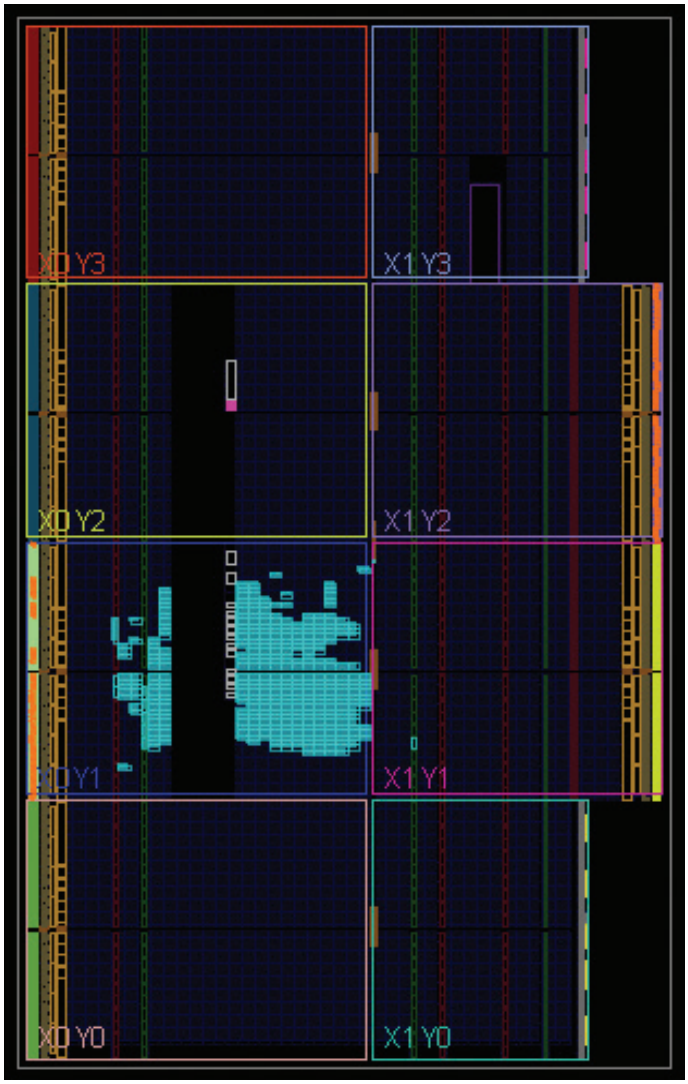


Figure 3. Vue du Floorplanner de Xilinx XCA100T (Alchitry Au+).

Listage 3. Extrait de rapport de Xilinx XCA100T.

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              | 2185 |     0 |     63400 |  3.45 |
|   LUT as Logic          | 2185 |     0 |     63400 |  3.45 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         | 1377 |     0 |    126800 |  1.09 |
|   Register as Flip Flop | 1377 |     0 |    126800 |  1.09 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    3 |     0 |     31700 | <0.01 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (monted@systemyde.com) ou contactez Elektor (redaction@elektor.fr).

Contributeurs

Texte : **Monte Dalrymple**
Rédaction : **Jan Buiting**
Traduction : **Yves Georges**
Mise en page : **Giel Dols**

PRODUITS

- **Livre en anglais de M. Dalrymple, « Inside an Open-Source Processor », Elektor 2021**
  Version papier : www.elektor.fr/19826
  Version numérique : www.elektor.fr/19827
- **Kit FPGA Alchitry Au**
  www.elektor.fr/19638