Journées Arinews

# Multiplieurs constants pour les FPGAs

Florent de Dinechin

Projet Arénaire,
École Normale Supérieure de Lyon

# Plan

- Introduction et motivations
- Multiplication par une constante par additions et décalages
  - ⋆ Un problème d'optimisation
  - ⋆ Recodage en chiffres signés
  - ⋆ Multiplication par une ou plusieurs constantes
- Le cas des FPGAs
  - ⋆ L'algorithme KCM
  - ⋆ JBits
  - ⋆ Une implantation de l'algorithme par recodage
  - ⋆ Comparaison et discussions
- Conclusion et perspectives

# Motivations

- Un multiplieur (ou additionneur) par une constante est bien plus petit (donc aussi plus rapide) qu'un multiplieur standard avec une des ses entrées fixées.
- Dans un FPGA, une constante peut changer...
- Mais il faut savoir produire la configuration pour la nouvelle constante assez vite.

# Multiplication par une constante par additions et décalages

On note $\ll$ l'opération de *décalage à gauche*.

$$
\begin{aligned}
2x &= x \ll 1 \\
3x &= x + x \ll 1 \\
4x &= x \ll 2 \\
5x &= x + x \ll 2 \\
6x &= (x + x \ll 1) \ll 1 \\
7x &= (x \ll 3) - x \\
8x &= x \ll 3 \\
&\dots \\
1587x &= x \ll 10 + x \ll 9 + x \ll 5 + x \ll 4 + x \ll 1 + x \\
&\qquad (1587 = \%11000110011) \\
&= x + x \ll 1 + (x + x \ll 1) \ll 4 + (x + x \ll 1) \ll 9
\end{aligned}
$$

# Un problème d'optimisation

*Quelle est la chaîne d'additions (ou soustractions) optimale pour multiplier $x$ (écrit sur $m$ bits) par la constante $k$ (écrite sur $n$ bits) ?*

$$\begin{array}{lcl} 1587x & = & x + x \ll 1 + x \ll 4 + x \ll 5 + x \ll 9 + x \ll 10 \\ & = & x + x \ll 1 + (x + x \ll 1) \ll 4 + (x + x \ll 1) \ll 9 \end{array}$$

La notion d'optimalité dépend bien sûr de la technologie visée :

- Dans un processeur, $+$ and $\ll$ sont des instructions machine (parfois fusionnées).
- En VLSI ou FPGA,
  - ⋆ $+$ utilise des ressources de calcul,
  - ⋆ $\ll$ utilise des ressources de routage,
  - ⋆ Les résultats intermédiaires peuvent être réutilisés (à un certain coût).
  - ⋆ La taille (en bits) des résultats intermédiaires croît de $m$ à $m + n$.
  - ⋆ Dans $S_1 + S_2 \ll i$ , les $i$ bits de poids faible sont ceux de $S_1$.

# Recodage en chiffres signés

L'idée (bien connue) est d'exprimer la constante dans la base $\{\overline{1}, 0, 1\}$
où $\overline{1}$ vaut $-1$.

Par exemple le nombre $0111$ $(= 1 + 2 + 4)$ sera recodé $100\overline{1}$ $(= 8 - 1)$.

Il existe un recodage canonique tel que :

- pour tout $k$, au moins un chiffre sur deux de son écriture est un zéro ;
- en moyenne, deux chiffres sur trois sont des zéros.

Il existe d'autres techniques de recodage plus performantes (Bernstein, Lefèvre).

# Multiplication par une ou plusieurs constantes

Cas fréquent (filtres) :
une même valeur $x$ sera multipliée successivement par plusieurs constantes.

Il est alors intéressant d'envisager une optimisation globale.

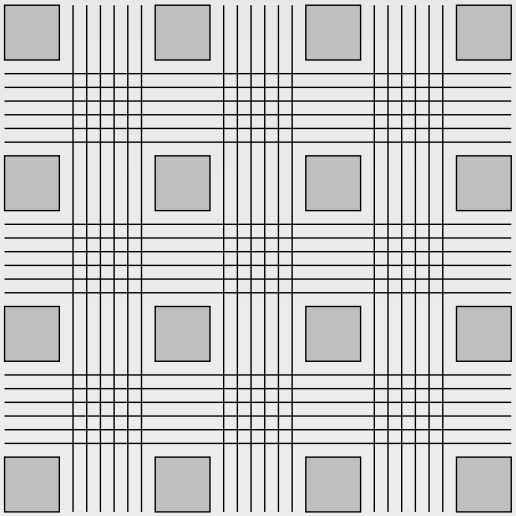
- Multiple Constant Multiplication [Mehendale et al]
- Distributed Arithmetics [Potkonjak et al]
- Multiple Constant Multiplier Trees [Benyamin et al]

Cela induit également un problème de routage global.

Nous ne nous intéressons (pour le moment) pas à ce problème.

# Le cas des FPGAs

- Structure physique :



- Outils de programmation hérités du VLSI
- Outils plus récents: PamDC, JBits.

# L'algorithme KCM [Chapman 94]

L'idée est d'exploiter la structure fine du FPGA (petites mémoires de $16 \times 1$ bits) en écrivant $x$ en base 16 :

$$x = \sum_{i=0}^{\lceil \frac{m}{4} \rceil} x_i.16^i \qquad \text{ce qui amène} \qquad kx = \sum_{i=0}^{\lceil \frac{m}{4} \rceil} kx_i.16^i = \sum_{i=0}^{\lceil \frac{m}{4} \rceil} (kx_i \ll 4i)$$

- Chaque $kx_i$ compte $n+4$ bits ($n$ est la taille de $k$)
- Chaque $kx_i$ est calculé par une table de $16 \times (n+4)$ bits, adressée par $x_i$.
- La somme est calculée par un arbre de profondeur $\log_2 \frac{m}{4}$. Par exemple :

$m = 8$ : $\underbrace{(kx_0 + kx_1 \ll 4)}_{n+8 \text{ bits}}$ (un additionneur de taille $n+4$)

$m = 16$ : $\underbrace{\underbrace{(kx_0 + kx_1 \ll 4)}_{n+8 \text{ bits}} + \underbrace{(kx_2 + kx_3 \ll 4)}_{n+8 \text{ bits}} \ll 8}_{n+16 \text{ bits}}$

$m = 32$ :

$$\underbrace{\underbrace{\underbrace{(p_0 \ + \ \underbrace{p_1 \ll 4)}_{n+8 \text{ bits}}}_{n+16 \text{ bits}} + \underbrace{\underbrace{(p_2 \ + \ \underbrace{p_3 \ll 4)}_{n+8 \text{ bits}}}_{n+16 \text{ bits}} \ll 8}_{n+32 \text{ bits}} + \underbrace{\underbrace{((p_4 \ + \ \underbrace{p_5 \ll 4)}_{n+8 \text{ bits}} + \underbrace{(p_6 \ + \ \underbrace{p_7 \ll 4)}_{n+8 \text{ bits}}}_{n+16 \text{ bits}} \ll 8)}_{n+16 \text{ bits}} \ll 16}_{n+32 \text{ bits}}$$

- Finalement, si l'on compte un *full-adder* par LUT, le coût matériel en LUT de KCM est

$$(\frac{m}{2} - 1)n + \frac{m}{2} \log_2 \frac{m}{4}$$

Cette formule correspond exactement aux tables fournies par Xilinx pour ses coeurs de multiplieurs KCM...

# Les nouvelles possibilités

Le placement et le routage de KCM ne dépendent pas de la constante $k$ :
seules les valeurs contenues dans les tables dépendent de $k$.

C'est tout ce que permettent les langages de description de matériel classiques (VHDL, Verilog) et le cycle de conception hérité du VLSI.

Les nouveaux outils (PamDC,JBits) visent les applications reconfigurables dynamiquement (filtre à coefficients changeant). Pour cela:

- la description de matériel consiste en des appels de bibliothèque dans un langage de programmation généraliste (C++, Java) ;
- la synthèse est une exécution du programme compilé ;
- cela permet une interaction matériel/logiciel dans le même langage, et des tas d'autres avantages (portabilité, OO, etc...)

# Les nouvelles possibilités pour les composants arithmétiques

Dans notre cas,

- le code d'un composant arithmétique va pouvoir inclure une quantité arbitraire d'optimisation ;
- cette optimisation pourra porter même sur la structure du composant.

Nous avons mis en œuvre un multiplieur constant utilisant le recodage canonique dont le placement et le routage dépendent de $k$.

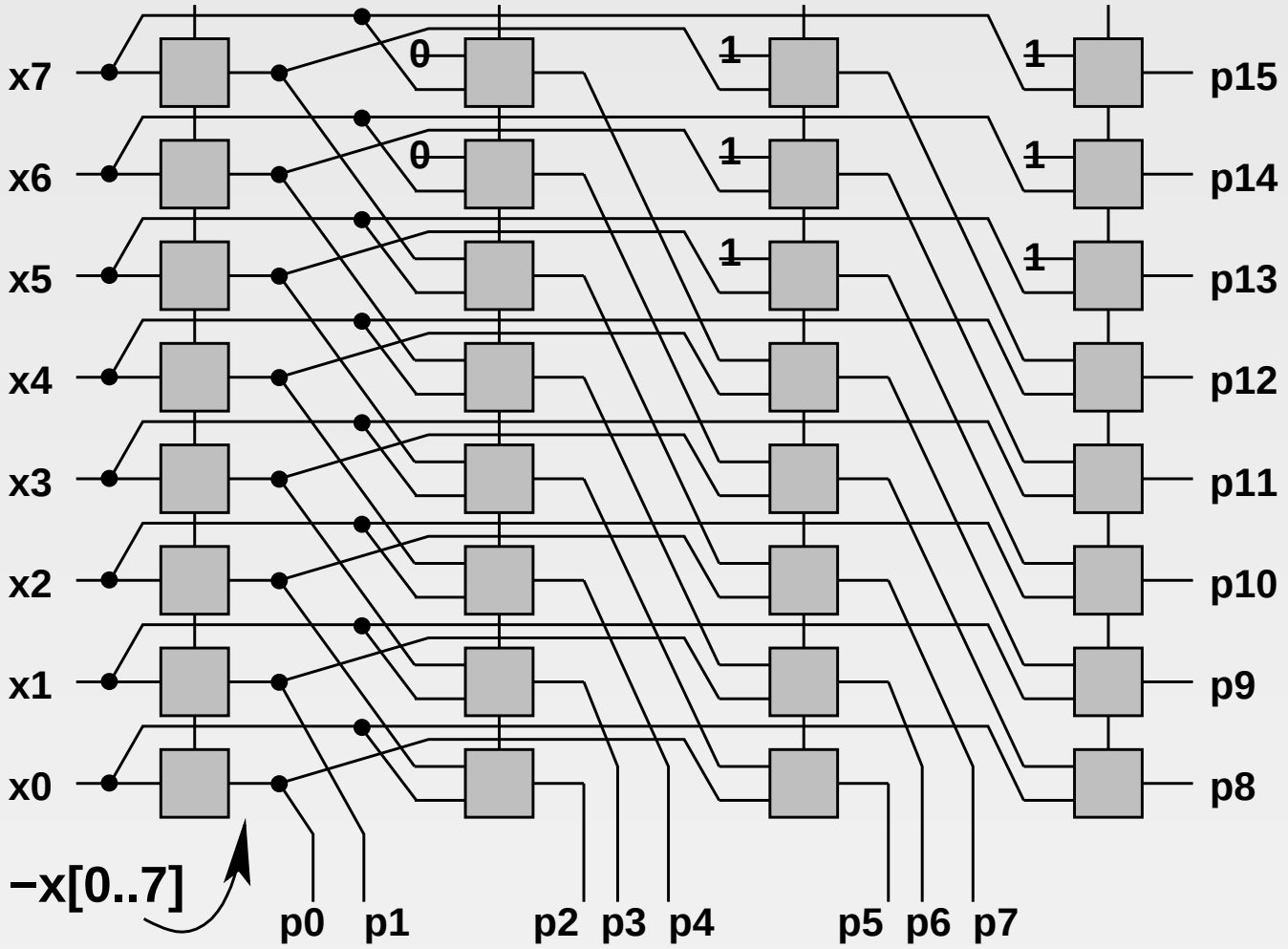
# L'implantation sur FPGA du multiplieur constant à recodage

- Exemple: $k = 221 = 100\bar{1}00\bar{1}01$

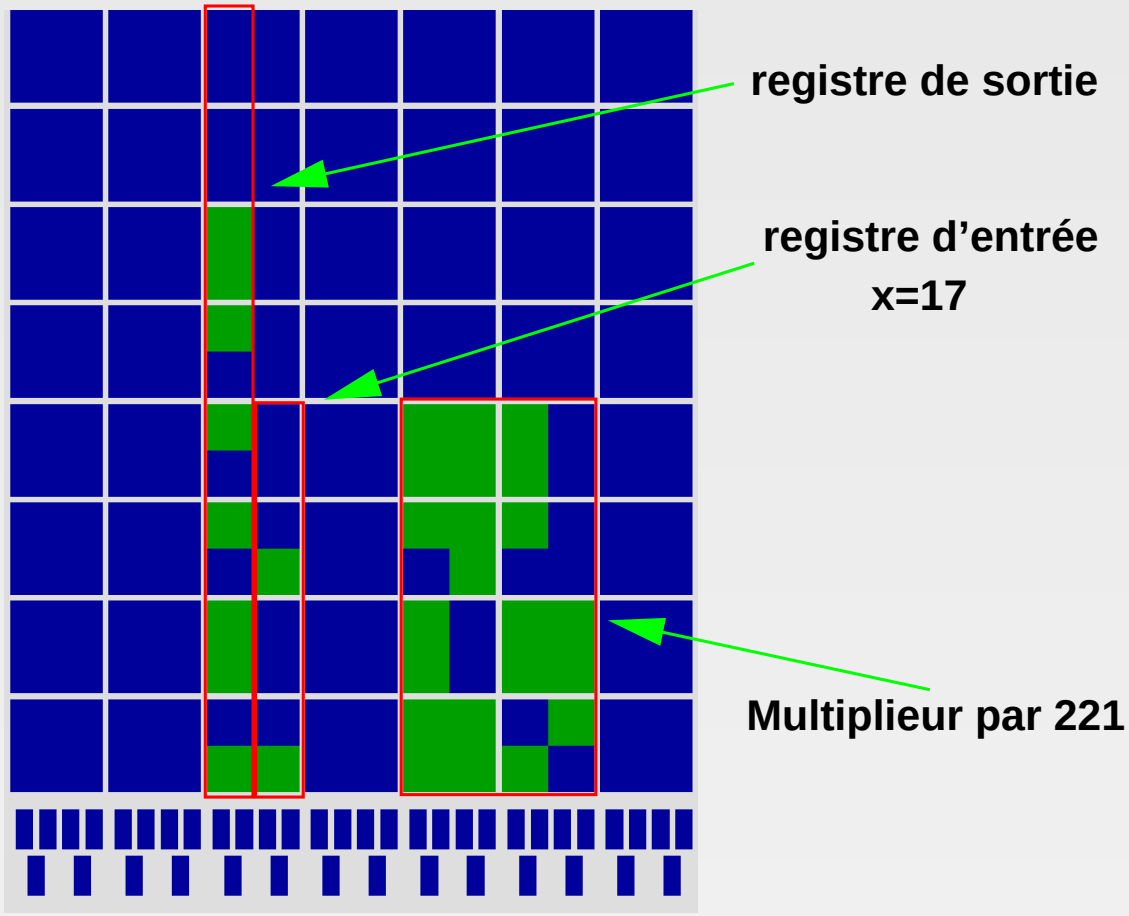
$$\begin{aligned} kx &= x \ll 8 - x \ll 5 - x \ll 2 + x \\ &= \underbrace{\underbrace{(\underbrace{(x - x \ll 2}_{m+2}) - x \ll 5}_{m+5}) + x \ll 8}_{m+8} \end{aligned}$$

- La taille de chaque additionneur est $m$ (la taille de $x$)...
- ... donc le multiplieur est un rectangle de hauteur $m$ et de largeur variable.
- Le coût matériel (en LUT) est au plus de $m \times \frac{n}{2}$, et en moyenne de $m \times \frac{n}{3}$.

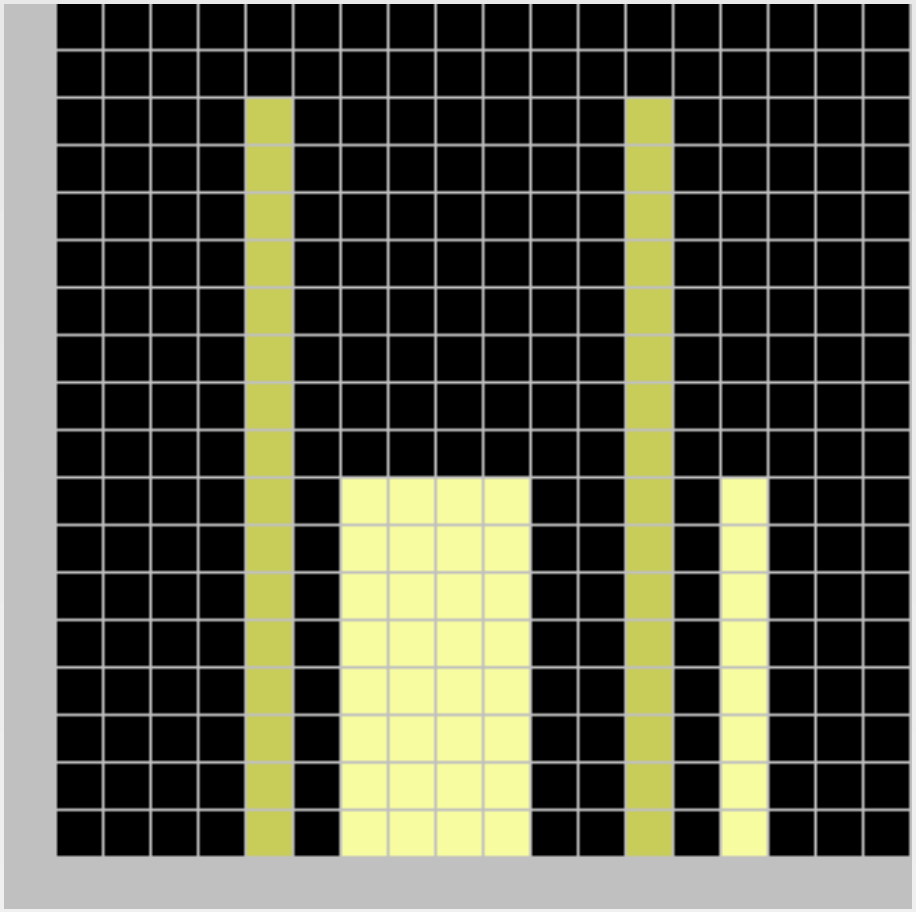
# Exemple : schéma de principe du multiplieur par 221

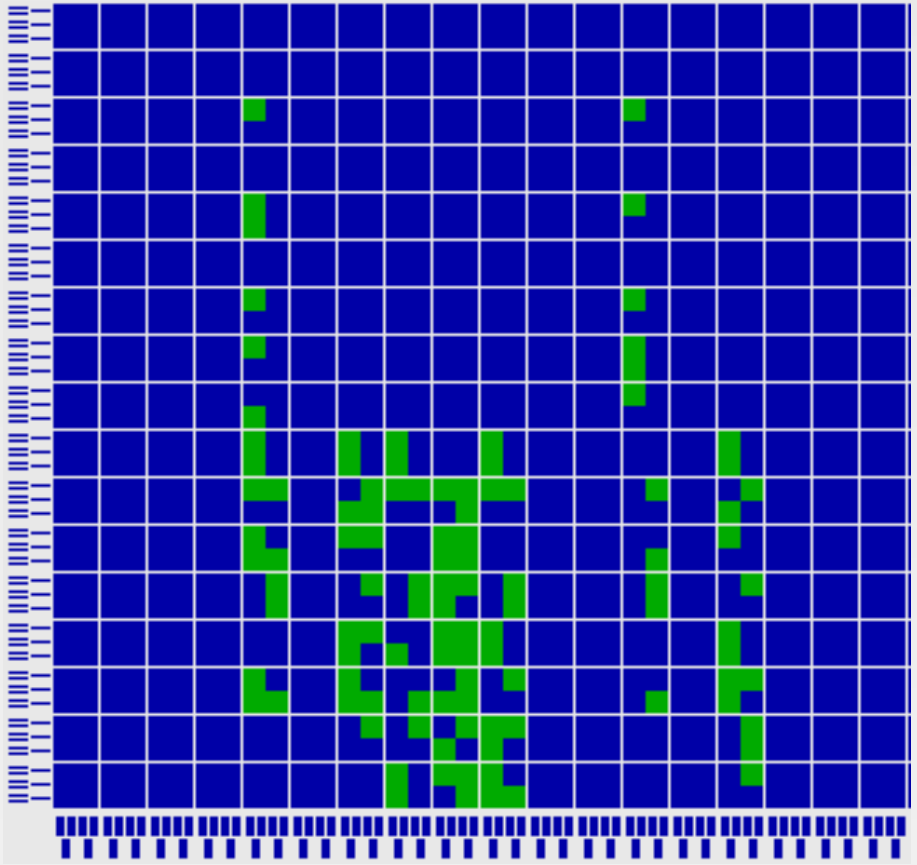


# Vue dans l'outil BoardScope du multiplieur par 221



# Deux multiplieurs par 58995 et 57344





# Quelques résultats

- En moyenne, un multiplieur $16 \times 16$bits occupe $24,2$ CLBs de 4 LUTs (test exhaustif).
- Le plus gros fait bien $4 \times 8$ CLBs...
- ... et le plus petit 0.

# Comparaison et discussion

\+ Notre multiplieur est toujours plus petit que KCM.

\− Nous n'avons pas comparé les vitesses mais il devrait être plus lent en moyenne (mais parfois bien plus rapide).

\= Les deux sont pipelinables facilement.

\− ... mais l'insertion du nôtre dans un pipeline plus vaste sera plus problématique.

\− Dans l'état actuel des outils de placement/routage, la taille variable est éliminatoire en ce qui concerne les applications à reconfiguration dynamique.

# Conclusion et perspective

- La suprématie d'un algorithme (ici KCM) peut venir de ce qu'il exploite au mieux des outils limités.
- JBits c'est pas mal.
- Le recodage canonique n'est pas intéressant. Il faut essayer des algorithmes d'optimisation plus complexes (Lefèvre).

  Mais étude de coût plus fine à faire
  (difficile d'obtenir un multiplieur rectangulaire).

# Questions?

- Introduction et motivations
- Multiplication par une constante par additions et décalages
  - ⋆ Un problème d'optimisation
  - ⋆ Recodage en chiffres signés
  - ⋆ Multiplication par une ou plusieurs constantes
- Le cas des FPGAs
  - ⋆ L'algorithme KCM
  - ⋆ JBits
  - ⋆ Une implantation de l'algorithme par recodage
  - ⋆ Comparaison et discussions
- Conclusion et perspectives