POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

INF3500 - Conception et réalisation de systèmes numériques

Laboratoire 3

Nicolas Deloumeau
Robin Gay

Automne 2024

Le calcul de la racine carré par la méthode de Newton

À la fin de ce laboratoire, vous devrez être capable de :

- Concevoir et modéliser en VHDL un chemin des données qui réalise des fonctions arithmétiques et logiques complexes au ni veau du transfert entre registres (RTL : *Register Transfer Level*). (B5)
  - Instancier des registres, dont des compteurs
  - Implémenter les fonctions arithmétiques et logiques pour les valeurs des registres, correspondant à une spécification par micro-opérations ou par pseudocode
- Composer un banc d'essai pour stimuler un modèle VHDL d'un chemin des données. (B5)
  - Générer un signal d'horloge et un signal de réinitialisation
  - Générer et appliquer des stimuli à un circuit séquentiel
  - Comparer les sorties du circuit à des réponses attendues pré-calculées
  - Utiliser des énoncés `assert` ou des conditions pour vérifier le module
  - Générer un chronogramme résultant de l'exécution du banc d'essai, et l'utiliser pour déboguer le circuit, entre autres pour résoudre les problèmes de synchronisation
- Implémenter un chemin des données sur un FPGA
  - Effectuer la synthèse et l'implémentation du circuit
  - Extraire, analyser et interpréter des métriques de coût d'implémentation
  - Programmer le FPGA et vérifier le fonctionnement correct du

  circuit avec l'interface PYNQ

Ce laboratoire s'appuie sur et complémente le matériel suivant :

1. Les procédures utilisées et les habiletés développées dans les laboratoires 0, 1 et 2
2. La matière des cours des semaines 4 (Modélisation et vérification de circuits séquentiels), 5 (Conception de chemins des données) et 6 (Conception et implémentation de fonctions arithmétiques sur FPGA).

Liste des fichiers

Fichier	Contenu
`division_par_reciproque.vhd`	Définit le module de division par réciproque dont il est question dans la partie « Implémentation de la division »
`src/racine_carree.vhd`	Contient le module de calcul de racine carré à compléter
`src/generateur_horloge_precis.vhd`	Contient un générateur d'horloge stable à fréquence variable qui permet de générer l'horloge pour la machine à états. *Vous n'avez pas à vous soucier de ce fichier*
`src/monopulseur.vhd`	Contient un monopulseur qui permet de compenser les effets de rebonds des boutons dus aux contacteurs et à l'appui humain. *Vous n'avez pas à vous soucier de ce fichier*
`src/utilitaires_inf3500_pkg.vhd`	Contient des définitions de fonctions utiles pour alléger le code et simplifier certaines opérations
src/racine_carree_tb.vhd	Contient le banc d'essai à compléter pour valider votre module
`src/top.vhd`	*Top-level* du projet
`xdc/PYNQ-Z2-v1.0.xdc`	Fichier de contraintes pour la PYNQ-Z2
`synth-impl/create_zynq_wrapper.tcl` `synth-impl/zynq.tcl`	Scripts de génération du *wrapper* pour le Zynq, à sourcer après avoir importé les sources

Partie 0 : Introduction

Vue d'ensemble

Dans cet exercice de laboratoire, on considère le problème de la conception d'un module qui calcule la racine carré X d'un nombre A, ($A = X \times X$). Ce module pourrait être ajouté à un microprocesseur pour en augmenter le jeu d'instructions.

La figure suivante illustre l'interface du module avec le monde extérieur. Les différents ports du module sont comme suit :

- Le nombre `i_A` est un entier non signé (nécessairement positif) exprimé avec N bits.
- La racine carré `o_X` est un nombre qui pourrait être fractionnaire, exprimé avec M bits : N / 2 bits pour la partie entière et (M - N / 2) bits pour la partie fractionnaire.
- Les calculs sont lancés quand le signal `i_go` a la valeur ‹ 1 › lors d'une transition positive du signal d'horloge `clk`. Le signal `o_fini` prend alors la valeur ‹ 0 ›.
- Le signal `o_fini` prend la valeur ‹ 1 › quand les calculs sont terminés, indiquant que le port `o_X` représente alors la racine carré de `i_A`.
- Le module est réinitialisé quand on place un ‹ 1 › sur le port `reset`.

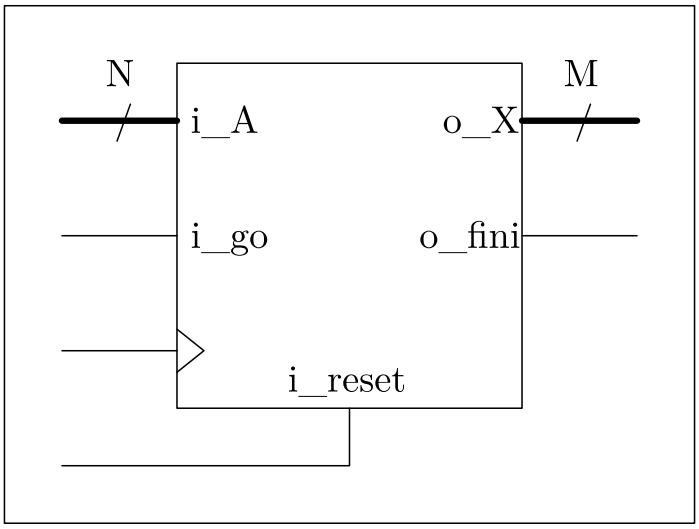

Fig. 1. – Module pour calculer la racine carré

Calcul de la racine carré par la méthode itérative de Newton

On peut calculer la racine carré $X = \sqrt{A}$ d'un nombre A par la méthode itérative de Newton.

À chaque itération k, on calcule :

$$X_{k+1} = \frac{X_k + \frac{A}{X_k}}{2}$$

Et la valeur de X_k converge vers $X = \sqrt{A}$ après quelques itérations.

Par exemple, pour A = 42871 et $X_0 = 255$, on obtiendrait la séquence montrée au tableau suivant en effectuant la division à l'aide d'une calculatrice et en arrondissant les X_k à l'entier le plus proche à chaque étape. La valeur finale après trois itérations, 207, est très proche de la racine carré qui est ~207.0531.

k	X_k
0	255
1	212
2	207

En pratique, et quand on n'a pas accès à une calculatrice, il y a plusieurs considérations à prendre en compte pour implémenter ces calculs avec un chemin des données :

- quelle précision choisir (combien de bits) pour représenter les calculs intermédiaires à l'intérieur du module;
- quelle valeur choisir pour X_0; et,
- comment implémenter la division.

Pseudocode et micro-opérations

Le calcul de la racine carré par la méthode itérative de Newton peut se modéliser avec le pseudocode et les micro-opérations suivantes. Ce pseudocode décrit une machine à états à deux états : {`attente`, `calculs`}.

```
si reset == '1' {
    etat ← "attente";
} sinon, répéter à chaque coup d'horloge {
    dans le cas où etat == "attente" {
        si go = '1' alors {
            k ← 0;
            A_int ← A;
            xk ← 255;
            etat ← "calculs";
        }
    }
    dans le cas où etat == "calculs" {
        fini ← 0;
        xk ← (xk + A / xk) / 2;
        k ← k + 1;
        si k = kmax alors {
            etat ← "attente";
        }
    }
}
X ← xk;
si etat = "attente" {
    fini ← 1;
} sinon {
    fini ← 0;
}
```

Pseudo-code 1. – Calcul de la racine carré par la méthode itérative de Newton

Implémentation de la division

La division générale n'est pas prise en charge par les outils de synthèse. Les diapositives de la semaine 6 présentent trois techniques synthétisables pour calculer la division générale. Pour ce laboratoire, on vous fournit un module de division générale par la technique de la réciproque dans le fichier `division_par_reciproque.vhd`. Ce module est similaire à celui présenté dans les diapositives de la semaine 6

Inspectez le code et les explications dans les diapositives pour en comprendre le fonctionnement.

Pour ce laboratoire, une valeur de W_frac de 14 bits semble produire de bons résultats. Vous pouvez expérimenter avec cette valeur et voir l'effet sur la précision des calculs et les coûts d'implémentation.

Partie 1 : conception du module de calcul de la racine carré et modélisation en VHDL

Complétez la modélisation du module de calcul la racine carré donné dans le fichier `racine_carree.vhd`.

Pour les besoins de ce laboratoire, vous devriez choisir `N` = 16, `M` = 8, `kmax` = 10, et `W_frac` = 14.

Attention : VHDL étant VHDL, le plus grand défi est peut-être dans le codage de l'opération xk ← (xk + A / xk) / 2.

- La division A / xk doit être produite par le module `division_par_reciproque.vhd` ou par un module de votre conception. Attention, ce module retourne un quotient sur 30 bits : 16 bits de partie entière et 14 bits de partie fractionnaire. Il faut enlever la partie fractionnaire et ne garder que les 8 bits les moins significatifs de la partie entière. Recommandation : faites-vous plusieurs exemples sur papier pour bien comprendre.
- L'addition de deux nombres de M bits produit une somme de M + 1 bits. Il faut calculer cette somme, la diviser par 2 selon l'opération, puis ramener la somme à M bits. Encore une fois, faites-vous des exemples sur papier. La fonction `RESIZE (ARG: UNSIGNED; NEW_SIZE: NATURAL) return UNSIGNED;` du package numeric.std peut simplifier l'écriture du code.

> **Évaluation**
>
> Modifiez le fichier `src/racine_carree.vhd` pour implémenter le calcul de la racine carré. *N'oubliez pas de mettre à jour votre dépôt Git.*

Partie 2 : banc d'essai

Vérifiez le fonctionnement de votre module `racine_carree.vhd` à l'aide du banc d'essai du fichier `racine_carree_tb.vhd`.

Bonifiez le banc d'essai pour bien vérifier le fonctionnement de votre module. Votre banc d'essai doit faire une stimulation exhaustive (avec tous les cas possibles de A) et le calcul de l'erreur. Il doit afficher l'erreur maximale et l'erreur moyenne.

> **Astuce**
>
> Commencez par vérifier quelques cas seulement. Il faut appliquer un nombre au port `i_A`, activer le signal `i_go`, attendre le bon nombre de coups d'horloge, puis inspecter la réponse.

> **Évaluation**
>
> Modifiez le banc d'essai pour vérifier votre module. *N'oubliez pas de mettre à jour votre dépôt Git.*

Partie 3 : implémentation sur la carte

Les fichiers suivants sont fournis pour aider à contrôler les interfaces de la carte et faire l'implémentation dans le FPGA. Ne les modifiez pas.

- `utilitaires_inf3500_pkg.vhd` : pour regrouper un ensemble de fonctions utiles pour les laboratoires du cours;
- `generateur_horloge_precis.vhd` : pour générer une horloge à une fréquence désirée à partir de l'horloge de la carte;
- `monopulseur.vhd` : pour synchroniser les actions des humains avec l'horloge du système;
- `top.vhd` : pour regrouper tous les fichiers lors de l'implémentation;
- `PYNQ-Z2-v1.0.xdc`, : pour établir des liens entre des identificateurs et des pattes du FPGA; et,

Inspectez le contenu du fichier `top.vhd` pour connaître le pairage entre les ports de l'entité `racine_carree` et les périphériques d'entrées et de sortie de la carte :

- Le port `reset` est relié au bouton 0;
- Le port `i_go` est relié au bouton 1;
- L'interface avec le Zynq permet d'entrer le nombre A dont on cherche la racine carré.
- Les boutons contrôlent ce qui est affiché.
- Le port `o_fini` est relié à la LED (0).
- La sortie `o_X` est reliée au processeur Zynq.

Faites la synthèse et l'implémentation de votre module pour votre carte. Vérifiez-en le fonctionnement.

> **! Évaluation**
>
> Ajoutez à votre dépôt Git votre *bitstream* final, sous `synth-impl/top.bit`.

Partie 4 : Défi - au-delà du A

Mise en garde. *Compléter correctement les parties précédentes peut donner une note de 17 / 20 (85%), ce qui peut normalement être interprété comme un A. La partie 4 demande du travail supplémentaire qui sort normalement des attentes du cours. Il n'est pas nécessaire de la compléter pour réussir le cours ni pour obtenir une bonne note. Il n'est pas recommandé de s'y attaquer si vous éprouvez des difficultés dans un autre cours. La partie 4 propose un défi supplémentaire pour les personnes qui souhaitent s'investir davantage dans le cours INF3500 en toute connaissance de cause.*

4a. Une meilleure estimation de X_0

On peut réduire de moitié le nombre d'itérations nécessaires pour que l'algorithme converge en choisissant une meilleure valeur de X_0 pour lancer l'algorithme. Pour $N = 16$ (avec A dans l'intervalle $[0, 65025]$), de bonnes valeurs de départ X_0 sont données au tableau suivant :

A	X_0
16384	255
4096	128
1024	64
256	32
64	16
16	8

! Évaluation

Modifiez votre module pour choisir une meilleure valeur de X_0. Expérimentez avec des valeurs réduites du nombre d'itérations pour voir l'effet.

4b. Modifier l'approche pour effectuer la division

! Évaluation

Proposez une autre manière d'implémenter la division et faites-en l'essai dans votre module. Vous pouvez vous inspirez des notes de cours, en particulier l'algorithme de Goldschmidt, ou proposer votre propre méthode. Le cas échéant, ajoutez votre *bitstream* sous `synth-impl/top_defi.bit`.

Remise

La remise se fait directement sur votre dépôt Git. Faites un *push* régulier de vos modifications, et faites un *push* final avant la date limite de la remise. Respectez l'arborescence de fichiers originale. Consultez le barème de correction pour la liste des fichiers à remettre.

Directives spéciales :

- Ne modifiez pas les noms des fichiers, entités ou architectures, ni les listes de `generics` ou de ports
- Remettez du code de très bonne qualité, lisible, bien indenté et commenté.
- Indiquez clairement la source de tout code que vous réutilisez ou dont vous vous inspirez.

Barème de correction

Le barème de correction est progressif. Il est relativement facile d'obtenir une note de passage (> 10) au laboratoire et il faut mettre du travail pour obtenir l'équivalent d'un A (17/20). Obtenir une note plus élevée (jusqu'à 20/20) nécessite plus de travail que ce qui est normalement demandé dans le cadre du cours et plus que les 9 heures que vous devez normalement passer par thème sur ce cours.

Critères	**Points**
Partie 1	
• Votre module final `racine_carree.vhd`	8
Partie 2	
• Votre banc d'essai complet dans le fichier `racine_carree_tb.vhd`	7
Partie 3	
• Votre fichier `top.bit` fonctionnel	2
Général	
• Qualité, lisibilité et élégance du code : alignement, choix des identificateurs, qualité et pertinence des commentaires, respect des consignes de remise incluant les noms des fichiers, orthographe, etc.	2
Sous-total	**17**
Partie 4	
• 4a : Meilleure estimation de X_0	2
• 4b : division modifiée	1
Total	**20**