INF3500 - Conception et réalisation de systèmes numériques

Labo 4 - Circuits séquentiels
18 février 2020

par Olivier Dion

Table des matières

1. Objectifs

Ce laboratoire sert à confirmer votre compréhension des circuits combinatoires. Le but sera donc d'apprendre à développer un circuit combinatoire synthétisable en **VHDL** avec **Vivado** . Pour ce faire, 4 objectifs sont à accomplir.

- Implémenter un circuit séquentiel.
- Comprendre le concept de machine à état.
- Vous familiariser avec la notion d'optimisation matérielle.
- Comprendre les dépendances entre les données reliées au cycle d'horloge.

Vous serez évalué sur ces objectifs ainsi que votre compréhension de l'exercice.

2. Conseils

Avant de commencer, voici quelques conseils.

- Utiliser **Git** (*GitBash* sur votre poste de travail) pour sauvegarder l'historique de vos travaux. Indexer seulement les fichiers code sources et autre fichier **non binaire**.
- Travaillez dans le dossier `C:\TEMP` et poussez vos travaux sur un serveur. La génération du *bitstream* sera plus rapide.
- Écrivez toujours le nom des autrices et matricules étudiant aux débuts de vos fichiers en commentaire. Ajoutez optionnellement une licence pour votre code source.
- La synthèse et l'implémentation sont des procédés assez longs. Imaginez un programme `C++` qui compile de 10 à 30 minutes. Relisez-vous pour vous sauver du temps !
- Dans le doute, consultez la **FAQ** du cours (`faq.pdf`).
- Écrivez vos rapports en `LaTeX`. La qualité de ceux-ci sera supérieure.
- Lisez le document en entier avant de commencer à travailler.

3. Contexte

Les architectures utilisant des circuits séquentiels sont très répandues dans le domaine des circuits numérique. Par exemple, les circuits séquentiels sont utilisés dans les processeurs à usage général, les processeurs graphiques, les modules spécialisés dans le cryptominage, les unités de calculs dans un système de contrôle d'un drone, etc.

De tels circuits numériques sont organisés en blocs de logique combinatoire séparés par de la logique séquentielle.

Par exemple, les machines à état, qui sont utilisées dans de nombreux circuits numériques, sont implémentées avec de la logique séquentielle, utilisées pour enregistrer l'état actuel dans une mémoire (une bascule), et avec de la logique combinatoire pour calculer la valeur de l'état futur.

4. Module SHA-256

Vous allez maintenant implémenter l'algorithme du `SHA-256` qui est à compléter dans le fichier `sha256.vhd`.

L'algorithme est expliqué en détail dans le fichier `fips180-4.pdf`. Les sections qui vous intéressent sont 2.2, 4.1.2, 4.2.2 et 6.2.2.

Pour résumé, l'algorithme est une machine à état. Cette machine à état prend en entrée un bloque ($M^{(i)}$) de 512 bits, qu'on va appelé **blob** (**B**inary **L**arge **Ob**ject) qui est composé de 16 mots de 32 bits et produit ce qu'on appelle un *digest*. On renvoie par la suite un *blob*, **sans** réanitiliser la machine à état, jusqu'à ce que tous les bits du message soient consommés. On rajoute un bit à 1, suivi d'un *padding* variable de 0 suivi de la longueur totale du message en bit sur 64 bits sur le dernier *blob*. Le dernier *digest* correspond au hachage final.

Pour ce laboratoire, vous n'avez pas à vous soucier des détails *blob*. Vous vous occupez simplement de la boucle principale de l'algorithme pour produire un nouveau *digest*.

Dans le document de référence à la section `6.2.2`, on dit de l'algorithme "For $i = 1$ to N". Le i fait référence au ième *blob* et au ième *digest*. Cela concerne surtout un module externe qui ferait plusieurs *digest* sur un long message. Donc, ne portez pas attention à l'indice i.

Aussi, pour que tout le monde soit au même niveau, on vous fournit les fonctions du `SHA-256` dans `functions-sha256-pkg.vhd`.

4.1 Avant de commencer

Il est conseillé de faire une machine à états sur papier avant de se lancer dans le code. Demandez à ce qu'on vienne valider votre machine à états !

4.2 Constantes du SHA-256

Dans l'algorithme de la prochaine section, on fait référence à un vecteur K. Ce vecteur comprend 64 valeurs sur 32 bits qui représentent les parties fractionnaires de la racine cubique des 64 premiers nombres premiers. Ce vecteur vous est donné dans le fichier `sha256.vhd` pour éviter les erreurs de copier/coller.

4.3 État de départ

Avant même de commencer le hachage du message, la valeur du digest est la suivante :

$$H_0^{(0)} = 0x6a09e667$$
$$H_1^{(0)} = 0xbb67ae85$$
$$H_2^{(0)} = 0x3c6ef372$$
$$H_3^{(0)} = 0xa54ff53a$$
$$H_4^{(0)} = 0x510e527f$$
$$H_5^{(0)} = 0x9b05688c$$
$$H_6^{(0)} = 0x1f83d9ab$$
$$H_7^{(0)} = 0x5be0cd19$$

NOTE ! Cet état de départ est celui d'un nouveau message, pas d'un nouveau *blob*.

4.4 Algorithme de digestion

ATTENTION ! N'oubliez pas que les signaux passant par une bascule vont être mis à jour 1 front d'horloge plus tard.

La figure 4.1 montre le pseudo-code pour la digestion, **sans** la logique du *reset* et de l'horloge. Notez que vous ne pouvez **pas** faire de boucle dans votre implémentation.

```
def digest(new_input, input, output, output_valid):

    if new_input:
        a = H[0]
        b = H[1]
        c = H[2]
        d = H[3]
        e = H[4]
        f = H[5]
        g = H[6]
        h = H[7]
        output_valid = False

    for i in range(0, 63):
        if i < 16:
            W[i] = input[i]
        else:
            W[i] = sigma3(W[i-2]) + W[i-7] + sigma2(W[i-15]) + W[i-16]

        T1 = h + sigma1(e) + ch(e, f, g) + K[i] + W[i]
        T2 = sigma0(a) + maj(a, b, c)
        h = g
        g = f
        f = e
        e = d + T1
        d = c
        c = b
        b = a
        a = T1 + T2

    H[0] += a
    H[1] += b
    H[2] += c
    H[3] += d
    H[4] += e
    H[5] += f
    H[6] += g
    H[7] += h

    output = H[0] & H[1] & H[2] & H[3] & H[4] & H[5] & H[6] & H[7]
    output_valid = True
```

FIGURE 4.1 – Pseudo-code de l'algorithme de digestion

Ce n'est pas synthétisable. Aussi, vous devez gérer la logique du *reset*, qui doit remettre l'état machine et la digestion à l'état initial.

Les fonctions fournies prennent et retournent des valeurs de type `unsigned`. Vous pouvez aussi utiliser vos propres fonctions faites dans les laboratoires précédents.

4.5 Livrable

Le fichier **sha256.vhd** avec l'algorithme de digestion correctement implémenté et synthétisable.

Votre machine à états à remettre dans votre rapport dans la description du système avec un **texte** l'expliquant.

5. Simulation

La simulation vous est donnée pour ce laboratoire pour que vous puisiez vous concentrer sur la conception de circuits séquentielle. Utilisez le fichier `sha256-tb.vhd` pour simuler votre circuit.

6. Implémentation

Faites l'implémentation du module `sha256-top.vhd` à l'aide du fichier de contrainte `top.xdc` et demandez à ce que l'on vienne vous évaluer.

Après l'implémentation, veuillez remplir le tableau suivant dans votre rapport et discuter des résultats. Consultez la FAQ (`faq.pdf`) au besoin.

LUT	Bascules	Fréquence (MHz)	Latence (ns)	Débit (opérations/s)

NOTE ! Pour calculer le débit, il faut compter le nombre de cycles d'horloge nécessaire pour partir de l'état *IDLE* et y revenir (on considère ici que le signal *new_input* est toujours à la bonne valeur pour changer d'état). On convertit par la suite le nombre de cycles en temps à l'aide de la période de l'horloge, soit 20 ns. Cela donne la latence. Montrez vos calculs dans votre rapport.

6.1 Livrable

Le fichier `sha256-top.bit`.

7. Questions

Ces questions sont à répondre dans votre rapport.

- (1.5) Lors de la synthèse, les boucles sont déroulées par le compilateur, si le nombre d'itérations est connu à la compilation, rendant ainsi le code synthétisable. Or, on vous a dit que vous ne pouvez pas faire de boucle pour votre module, car ça ne serait pas synthétisable. Expliquez pour **quelle** raison majeure cela est le cas.
- (0.75 x 2) Les modules *debouncer* et *pulse* (`debouncer.vhd`, `pulse.vhd`) sont deux modules séquentiels. Pour chaque module, énumérer le(s) bascule(s) et expliquer ce que le module fait.

8. Rapport

Vous devez écrire un rapport selon les directives dans le fichier `rapport.pdf`.

Seuls les fichiers de type *pdf*, *DjVu* et *ps* sont acceptés pour le rapport. Aucun fichier de type Word (doc, docx) n'est accepté.

Assurez-vous d'inclure tous les fichiers listés dans les sections `Livrable(s)`.

9. Barème

Critères	**Points**
Module SHA-256	8
Implémentation	7
Questions	3
Rapport : Présentation et qualité de la langue	2
Total	20