**Travail Pratique #2**

INF3500 : Conception et réalisation de systèmes numériques

Rapport remis par :

Anastasiya Basanets (1933929)

Mariam Sarwat (1928777)

Groupe : B1

|  |  |
| --- | --- |
| Critère | Points |
| Partie A : conception du module  Partie B : banc d’essai et simulation  Partie C : implémentation  Partie D : analyse et Discussion  Rapport : présentation et qualité du français | 8  4  4  2  2 |
| Total | 20 |

École Polytechnique de Montréal

Date de remise (25-02-2019)

**Partie A : Implémentation de la boucle principale de l’algorithme SHA-256**

Tout d’abord, nous avons dû écrire notre pseudocode en VHDL. Pour ce faire, nous nous sommes inspiré des exemples vus en cours et nous avons réalisé une machine à états. La machine à état contient les états suivants :

État INIT : Dans cet état, on initialise les valeurs de départ dans H.

État 1 : Dans cet état, on vérifie si l’entrée est différente de W et on initialise les variables dans nos signaux.

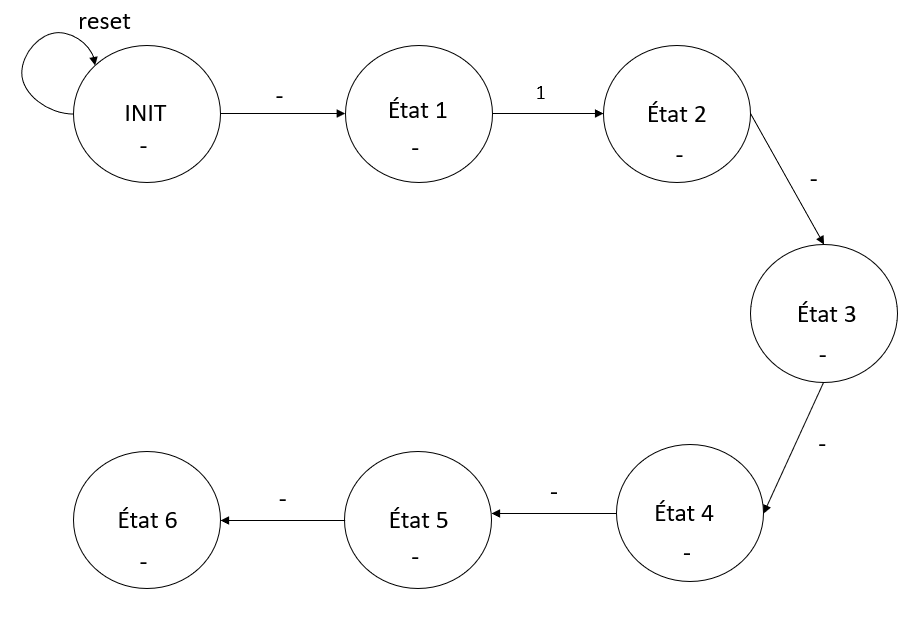
État 2 : On effectue la boucle principale de T1 = h + Σ {256} 1 (e) + Ch (e, f, g) + K + W jusqu’à f = e.

État 3 : On effectue e = d + T1.

État 4 : On effectue la boucle principale de d = c jusqu’à a = T1 + T2.

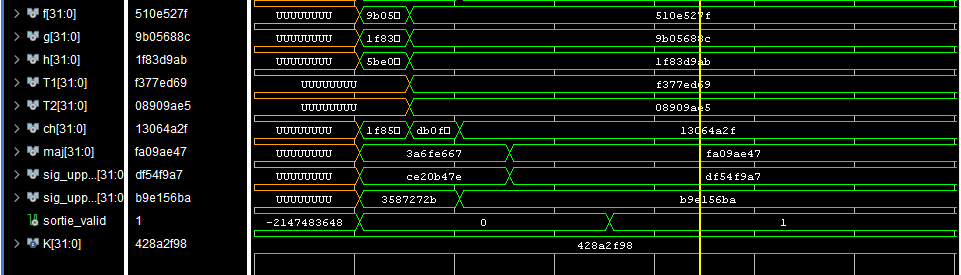
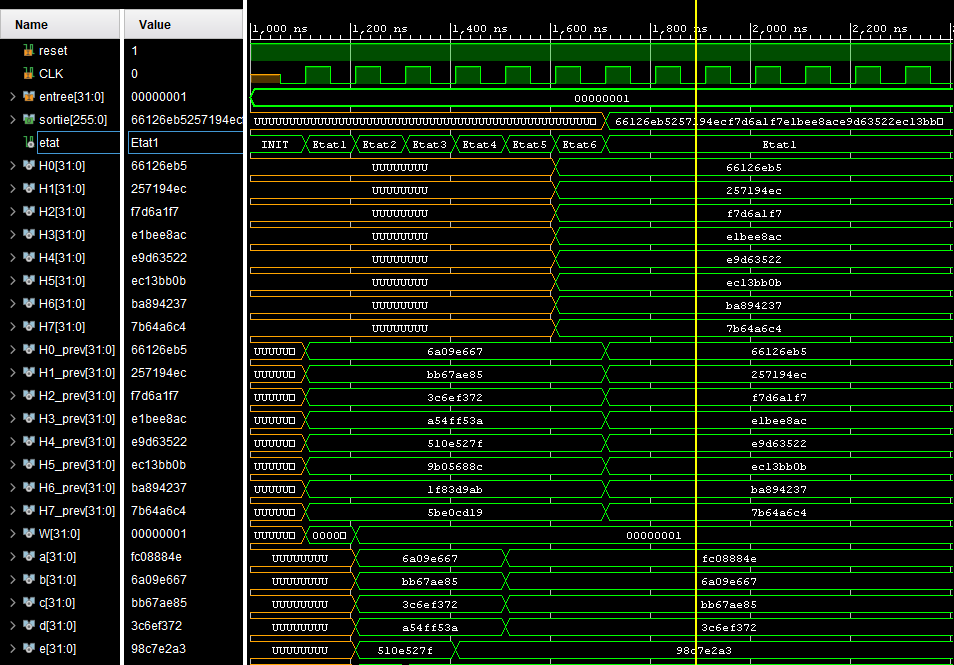
État 5 : On met à jour le H.

État 6 : On concatène H dans la sortie.



Par la suite, nous avons simulé nos fonctions selon les étapes fournies dans TP.

*Schéma 1 :* Simulation de notre machine à états

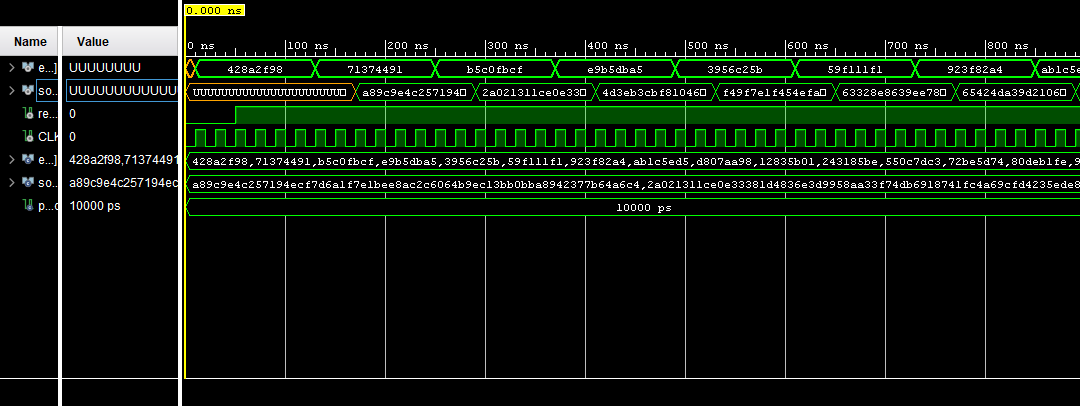


Dans cette simulation, on s’aperçoit que, de l’état INIT à 6, on initialise les valeurs de H (H\_prev). De plus, on initialise nos valeurs signaux (ex : a, b, c…). Rendu à l’état 6, on s’aperçoit que la sortie a été calculée en utilisant les valeurs de H0…H7. Par la suite, on met à jour nos valeurs de H\_prev. Finalement, on retombe dans l’état 1, puisque l’entrée est différente de l’entrée W. Ce qui veut dire que notre système va recalculer une nouvelle sortie en fonction de la nouvelle entrée lors du prochain clock.

**Partie B : Simulation**

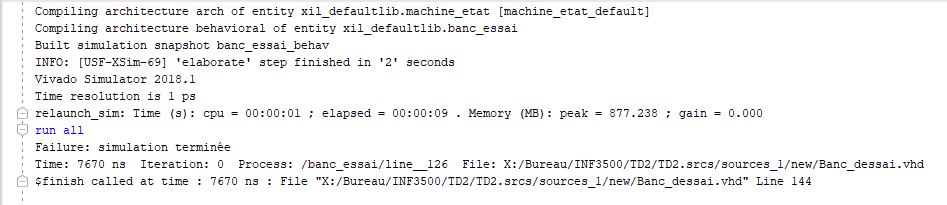
Pour effectuer la simulation, nous avons, dans un premier temps, écrit le pseudocode en C++. Cela nous a permis de générer des sorties avec des entrées choisies. Ces sorties nous ont été utiles pour la simulation. En effet, nous les avons insérés dans notre banc d’essai. Cette simulation nous a permis de vérifier si les valeurs de sorties générées par notre code C++ sont identiques à celles en VHDL.

*Schéma 2 :* Simulation de notre banc d’essai



On peut observer que les valeurs de sorties, correspondent bien aux valeurs d’entrées de notre banc d’essai et de notre code C++.

*Schéma 3 :* Simulation complétée du banc d’essai



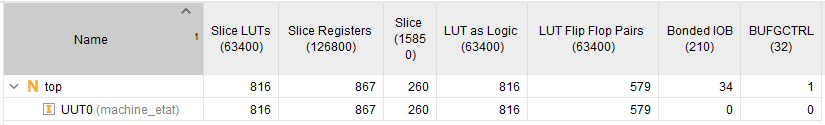
Sur la photo présente, on peut observer que la simulation est terminée et n’affiche aucune erreur. Cela signifie que les valeurs trouvées en sorties correspondent pour les deux codes. On peut donc conclure que nos deux codes sont réalisés de manière convenable.

**Partie C : Implémentation**

Afin de faire l’implémentation des fonctions de compressions nous avons dû créer un fichier top qui les relient ensembles dans l’ordre décrit dans l’énoncé du TP2. Il était important de s’assurer que notre valeur d’entrée et de sortie étaient en 16 bits. Alors, pour ce faire, nous avons fait la conversion de 16 bits à 32 bits pour notre entrée et de 32 bits à 16 bits pour la sortie. Lorsque la conversion est complète, la valeur d’entrée va passer par tous les fonctions dans notre fichier top et va sortir une valeur de sortie. De plus, nous nous sommes assuré d’inclure les signaux le clock et reset pour que l’implémentation du code puisse s’exécuter convenablement. Nous avons réussi à générer le bitstream.

**Discussion**

*Schéma 5 :* Tableau des résultats





|  |  |  |  |
| --- | --- | --- | --- |
| LUT | Bascules | Fréquence | Latence |
| 816 | 579 | 100 MHz | 10 ns |

On observe que notre code a recourt à 816 LUT et 579 bascules. De plus, la fréquence et la période a été fixé dans le fichier de contraintes.

Finalement, nous avons atteint les objectifs de ce laboratoire, soit d’implémenter la boucle principale de l’algorithme SHA-256. De plus, nous avons réussi à simuler et implémenter correctement notre circuit.