Sprint 9 : User-stories

auteur : Guillaume Leinen

Hussein : Comment calculer un cos en VHDL (grace au HLS) => implémentation d'une table LUT sur la carte.

Difficultés rencontrées

Difficulté de compréhension de l'utilité des Tables LUT, et de l'application à notre projet.

travail implémenté

Pas de code disponible pour un cos à travers une bascule LUT.

Guillaume : Huffman en vhdl => comparaison avec huffman c sur riscV

Difficultés rencontrées

Huffman nécessite de définir des types non natifs (noeuds, dictionnaires, etc...) ce qui n'est pas possible via la HLS (c'est plutôt la nécessité d'affecter statiquement la mémoire aux variables, ce qui empêche la création d'un tableau/dictionnaire d'encodage (taille variable) . Il reste encore une possibilité pour effectuer cela en vhdl natif mais cela est mis au second plan pour l'instant. Une fonction traduisible relativement facilement est la DCT, ce qui tombe bien pour les comparatifs de performance puisque c'est le "bottleneck" du code en go.

Travail implémenté

L'encodage de Huffman n'est pas implémenté en raison des difficultés énoncées plus haut. En revanche, on a trouvé dans le même temps des moyens de créer une LUT utile au calcul du cos de la DCT. La piste de la library std_arithm en VHDL est également envisagée.

Hugo : traduire zigzag en go et bitstream

difficultés rencontrées

Le code fonctionnait mais n'est pas correct algorithmiquement parlant. Grâce aux tests unitaire nous sommes en mesure de trouver ces erreurs et nous avons corrigé le zigzag. Nouvelle syntaxe du go à prendre en main. Bitstream assez difficile à "parser", cela a nécessité un schema UML des trames de bitstreams afin de bien comprendre quand s'arrêter et quel champ filtrer.

travail implémenté

Zigzag scan fonctionnel en go. Bitstream en WIP, il manque l'ajout des entêtes, nécessaire à la transmission du dictionnaire.

Jean-Noël : Se procurer une carte RBPi et jouer avec rasbian (helloworld ou plus si affinité)

difficultés rencontrés

Configuration de la Raspberry pi. se procurer le matériel et les périphériques nécessaire à l'initialisation de la carte.

travail implémenté

Jean-Noël peut se servir d'une RBpi entièrement configuré niveau SSH, avec accès à internet vie le pc host.

Alexandre: Terminer la chaine d'encodage en GO

difficultés rencontrés
Temps imparti, difficile de trouver du temps en ce moment. la Syntaxe et le processus de debug et profiling beaucoup plus simple en go qu'en c.

travail implémenté
la chaîne d'encodage en go est fini à l'exception du bitstream. le récepteur est capable de recevoir de l'information. un profiling du code a été réalisé, il met en lumière l'importance de la DCT dans la complexité temporelle de l'encodage. L'encodeur est accelerable via des astuces de calculs (dev limité de taylor pour cos, programmation en concurrence, etc...)

Tout le monde : faire un hello world en go.

Hello word effectué par tout l'équipe (grâce a Gobyexample).

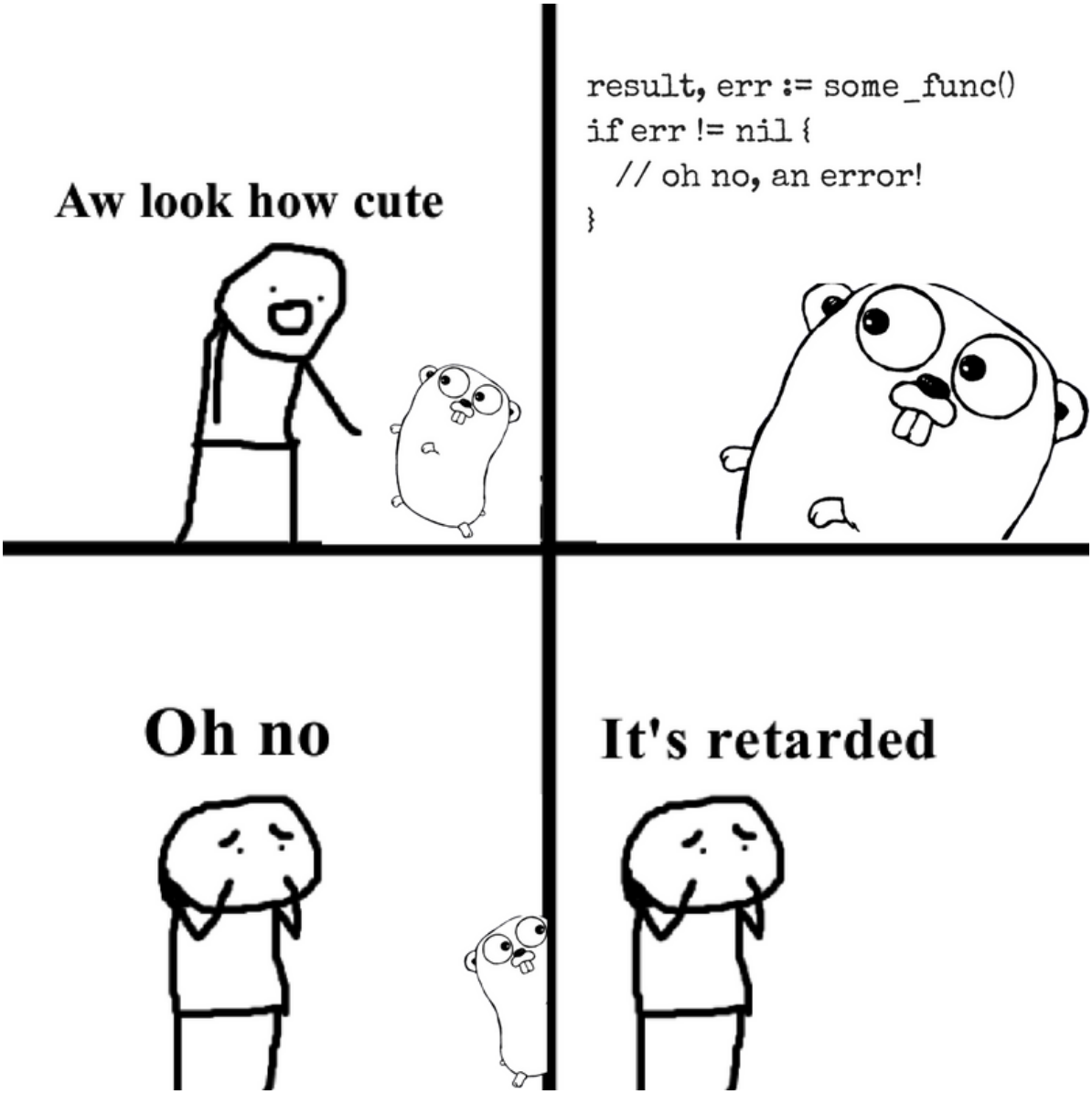
impressions générales :

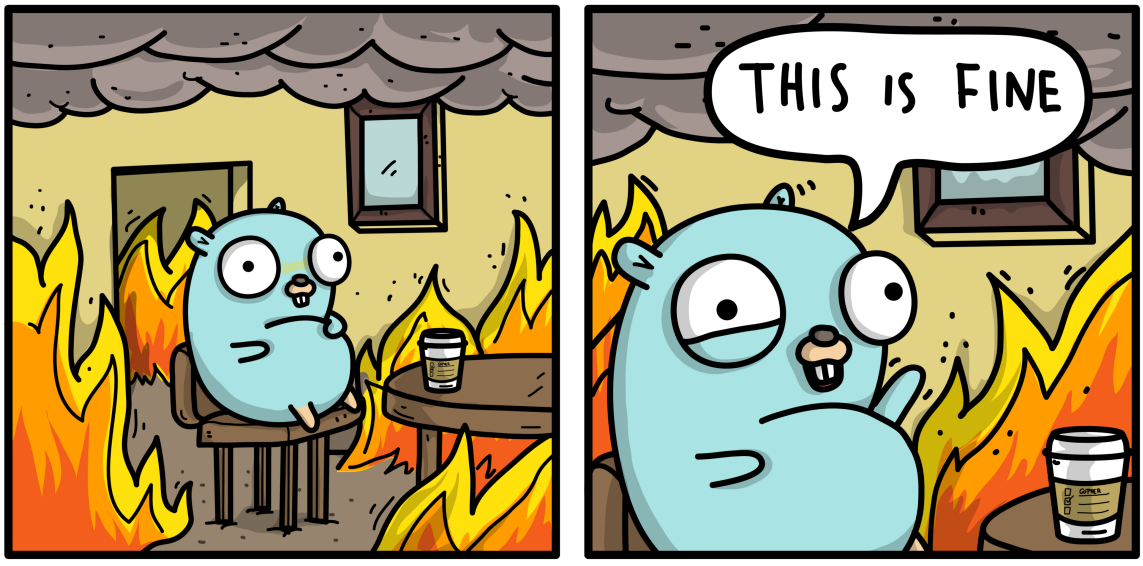
- la syntaxe est plus facile à lire que le c, notamment la gestion des array (slice) et la définition de méthode de "struct" se rapprochant de la POO
- Garbage collector intégré au langage (le go donne la localisation d'un erreur et est beaucoup plus verbeux sur le debug que le c et GCC)
- Outils de profiling du code intégré et bibliothèque de tests unitaires intégré.
- Bcp de garde fou : il n'y a pas de runtime error, juste des compilations d'erreurs
- Cross-compilation existante pour le langage.

inconvénients:

- Taille du binaire conséquente (un hello world de 2Mo ça fait mal)
- cross-compilation impossible en riscV32 bits, ainsi que la HLS via vivado.

Suite : Vacances ...


Aw look how cute
result, err := some_func()
if err != nil {
// oh no, an error!
}
Oh no
It's retarded


THIS IS FINE