Rapport d'avancement mi-projet

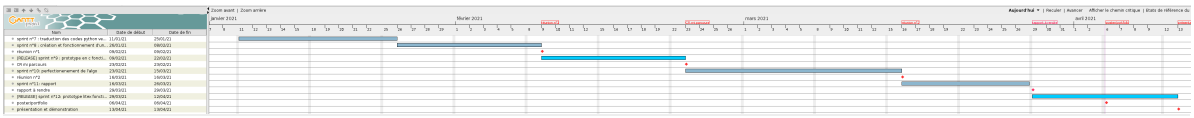
Présents : intégralité de l'équipe EVEEX, Jean-Christophe Le Lann, Pascal Cotret

Absent : Joël Champeau.

Points à aborder durant la réunion

- avancement par rapport au diagramme de Gantt
- développement et tests réalisés
- énumération des taches restantes et affectation

Avancement du projet par rapport à la prévision



Projet UE 4.4 phase 2

Tâches

Nom	Date de début	Date de fin
sprint n°7 : traduction des codes python vers c	11/01/21	25/01/21
sprint n°8 : création et fonctionnement d'un SOC litex	26/01/21	08/02/21
réunion n°1	09/02/21	09/02/21
[RELEASE] sprint n°9 : prototype en c fonctionnel	09/02/21	22/02/21
CR mi parcours	23/02/21	23/02/21
sprint n°10: perfectionenement de l'algo	23/02/21	15/03/21
réunion n°2	16/03/21	16/03/21
sprint n°11: rapport	16/03/21	26/03/21
rapport à rendre	29/03/21	29/03/21
[RELEASE] sprint n°12: prototype litex fonctionnnel	29/03/21	12/04/21
poster/portfolio	06/04/21	06/04/21
présentation et démonstration	13/04/21	13/04/21

Sprint 7 : traduction du code python vers le c

Réalisation : Nous avons créés des types non natifs en c, qui nous permettra d'implémenter les fonctions métiers du prototype python. La plupart des fonctions ont été indépendament dévellopés, cela comprend le socket, l'encodage de huffman, la DCT, etc... . Il reste à coder les liens entre les différents blocs de traitement.

JCLL : On a une matrice de nombres avec des opérations à effectuer dessus, pourquoi faire si compliqué ? J'ai l'impression que le projet s'est égaré.

Notre point de vue : Le code est devenu assez compliqué avec le temps, nous en sommes à environ 3000 lignes de code uniquement en c. Il va etre quasi impossible de tout redevelloper en vhdl, et il serait dommage d'abandonner le code tel-quel vu le temps passé dessus. Cela va nécessité une réorientation du projet.

Solution potentielle => développer Eveex sur raspberry pi en C/go (cross-compilation très simple) et en parallèle passer des fonctions de l'algorithme sur Vivado HLS. Il y a des expériences complémentaires à faire en vivado. l'interet serait de comparer la vitesse de traitement entre software et programmation matérielle pour différents blocs de traitements.

Sprint 8: création et fonctionement d'un SOC RiscV

Nous sommes en capacité de reproduire un SOC d'architecture RiscV au sein d'un FPGA. Ce SOC est en 32 bit et permet nottament de lui introduire un noyau Linux optimisé appelé Buildroot. On peut ensuite intégrer au Buildroot les paquets necessaire à notre programme, tout en gardant une taille raisonable (le noyau de base fait 7 Mo).

On va mettre de coté Buildroot pour l'instant pour se concentrer sur une compilation ARM et du VHDL natif.

Dévellopement réalisé depuis la fin du Semestre 3

- python : Une version stable de EVEEX en python a été packager. il suffit à l'utilisateur d'installer le paquet et EVEEX est installer avec toutes les dépendances python. L'algorithme est maintenant capable d'encoder une vidéo, mais pas encore de la décoder.
    - performances : le fichier compressé est 2 fois plus lourd que MJPEG (22Mbits pour 3 secondes en 480p), compression lente (non temps réel) => 0,46 Fps (frame par seconde) pour du 480p.
    - JCLL : c'est bien, il faut valoriser les performances de l'algorithme.
- c : 1ère étape de structuration avec recherche des types à implementer => Avec le recul, faire une traduction bête et méchante n'a pas été la meilleure idée à priori. Le code est devenu trop lourd/hors de contrôle, on dépasse les 3000 lignes. Fuites de mémoire difficiles à colmater (2 semaines). Les blocs métiers fonctionnent indépendamment mais il reste 1 ou 2 liaisons à developper.
    - JCLL : Il faudra argumenter ça au rendu, pourquoi avoir choisir une orientation objet en c ?
    - Réponse : *On a choisi le c pour 3 raisons : la première etait la compatibilité avec Vivado Hls, qui est possible avec du C ou du C++. La 2ème raison c'est les performances car le language est bas niveau. Enfin la dernière était que l'on a eu des cours de C au semestre 3 (on ne connaissait pas tous le c++/go).
- vhdl : On peut construire un SOC riscV sur le FPGA, et compiler du code pour l'architecture riscV afin d'executer le code sur le FPGA. Un des problèmes va etre le traitement des données d'entrées de la camera : un SOC riscv cadencé à 100mhz ne sera pas assez performant pour capturer le flux vidéo brut provenant de la camera. Il va donc etre nécessaire de traiet la caméra séparement, ce qui est contre l'interet de LiteX.
    - JCLL : Il faut donc se réorienter. Pourquoi pas dévelloper certaines fonctions du code c vers le vhdl en passant par la HLS ou en développant à la main, et ainsi comparé la vitesse d'execution du software a celle d'une implémentation hardware.

Suite du travail

La réorientation du projet est donc la suivante => on a un algorithme maison que l'on maitrise.

On va d'abord réécrire EVEEX en Go (le code Go est beaucoup plus facile a appréhender que le c donc nous redeveloppons EVEEX en go, par contre il n'est pas possible de compiler du go pour une architecture riscv 32 bits, le c restera là pour l'optimisation des blocs de traitements et l'implémentation riscV).

Ensuite on va s'interesser à différentes approches de l'embarqué :

- 1ere approche (la plus simple) => Cross-compiler EVEEX vers une architecture ARM plus optimisée que l'architecture x86 des ordinateurs actuels, et exporter le binaire vers des cartes raspberrypi, premierement sur raspbian puis potentiellement sur un linux Buildroot pour l'optimisation. Il devrait etre possible de réaliser un démonstrateur sur cette plateforme en temps voulu : 2 cartes RBPi, une avec une caméra et une avec un écran, qui communique en réseau avec une carte encodeuse et une carte décodeuse.
    - inconvénient : ARM n'est pas open-source.
- 2ème approche => L'architecture RiscV est open-source et gratuite à l'utilisation => il serait interessant de porter plusieurs blocs de traitement de EVEEX vers un SOC riscV sur une carte FPGA, afin d'en évaluer les performances. La perspective d'avoir une chaine d'encodage fonctionelle en entier est faible compte tenu des performance limités du processeur "émulé" sur la carte fpga. Une autre possibilité serait l'émulation d'un SOC riscV sur PC gràce à QEMU, nottament pour le débug. Nous sommes en train d'étudier la question.
- 3ème approche => les FPGA et le VHDL directement. Il serait interessant de dévelloper des blocs directement en VHDL grace nottament à des outils de HLS comme Vitis de Vivado ou encore depuis Ruby gràce à ruby_rtl dévelloper par M.Le Lann, toujours à des fins comparatifs en terme de performance.

Ces recherches nous permettrons de présenter les avantages et inconvénients de 3 technologies de systèmes embarqués. De plus l'implémentation ARM nous permettra probablement d'avoir un démonstrateur fonctionnel pour Avril.

Point méthode Agile

M.Champeau est injoignable....

Le travail dans le groupe est toujours sérieux, même si cela devient de plus en plus compliqué d'etre motivé. Il réside encore des interrogations nottament sur la tenue ou non d'un forum en présentiel afin de présenter le démonstrateur.

Revoir la durée des Sprints : faire plus court et de taille variable. (sprints actuels de 2 semaines)