Report HW Virtualization and trust
Ariane ODONI, Maia ROCA, Kylie WU - OCC3

Dans le cadre de notre projet HAL, nous avons suivi une progression par étapes pour développer les fonctionnalités GPIO, USART, SPI et I2C sur plusieurs plateformes. Voici un résumé de nos réalisations et défis :

1. Plateforme Arduino

Nous avons commencé par implémenter toutes les fonctionnalités sur une Arduino Uno. Grâce à une organisation structurée et des tests sur matériel physique, nous avons obtenu les résultats escomptés pour le GPIO, l'USART, et l'I2C. Cependant, pour le SPI, bien que la LED s'allume comme prévu, elle ne clignote pas. Nous pensons que le problème vient de l'usage d'un mauvais pin mais, afin de ne pas perdre plus de temps (malgré de nombreuses tentatives), nous avons décidé de nous concentrer sur d'autres aspects du projet.

2. Plateforme Cortex, Raspberry Pi

Nous avons tenté d'implémenter notre HAL sur un Cortex-M7 et Raspberry Pi, mais nous avons rencontré des obstacles techniques, nous n'arrivions pas à faire en sorte que le Cortex ou le Raspberry pi fonctionnent ensemble avec le premier target. Même si nous avons essayé de simuler le Cortex avec Qemu, les commandes n'ont pas renvoyé de résultats satisfaisants. Ces problèmes nous ont conduits à explorer d'autres options.

4. Plateforme ESP32

Nous avons ensuite travaillé sur le partage de notre code sur l'ESP32. Le code compile correctement, mais faute de matériel physique disponible pour effectuer les tests, nous n'avons pas pu valider cette étape. Nous avons tenté plusieurs fois d'utiliser Qemu également pour les simulations mais nous ne sommes pas arrivées à le linker avec notre projet (bien que ça ait fonctionné pour le projet test du tutoriel et que toutes les installations tierces nécessaires à son fonctionnement aient aussi été faites).

5. Outils et organisation

Au début, nous avions une structure de dossier inadéquate. Après avoir ajusté cette organisation, nous avons décidé de procéder étape par étape, en validant chaque fonctionnalité d'abord sur l'Arduino. Cette méthodologie nous a permis de minimiser les erreurs et de progresser efficacement.

6. Explication de la structure de notre code

Nous allons maintenant préciser comment notre code s'organise, nous avons dans le dossier source :

- un **dossier atmega** avec un fichier pour chaque code correspondant au gpio, à l'i2c, au spi et à l'usart qui fonctionnent sur notre Arduino

- un **dossier esp32** avec un fichier par code correspondant au gpio, à l'i2c, au spi et à l'usart pour notre esp32
- un **fichier atmega.rs** qui permet de lancer l'utilisation des différentes fonctionnalités de l'Arduino (gpio, usart, spi…)
- un **fichier esp32.rs** qui a la même fonction que le fichier précédent mais pour l'esp32.
- un **fichier main.rs** qui permet de lancer soit le fichier esp32.rs soit le fichier atmega.rs.

Pour un peu plus de précisions, notre fichier main.rs lance le fichier atmega.rs lorsque nous utilisons la première commande :

cargo +nightly build -Z build-std=core --target avr-unknown-gnu-atmega328 --release --features atmega

et le fichier esp32.rs lorsque l'on utilise la commande :

cargo +nightly build -Z build-std=core --target avr-unknown-gnu-atmega328 --release --features esp32

Si nous voulons que le programme se redirige vers la bonne feature directement, nous devons enlever le commentaire de la ligne #default = ["atmega","esp32"] du fichier Cargo.toml et la ligne cfg_if::cfg_if! dans le main.rs. Nous n'avons pas pu tester cette fonctionnalité car nous n'avions pas d'esp32 à notre disposition et que nous ne sommes pas arrivées à le faire fonctionner avec Qemu.

Un autre fichier que nous avons modifié est notre Cargo.toml dans lequel nous avons :
- nos **dependencies** : avr device
- nos **features** : atmega et esp32
- notre **build target** : atmega328

Conclusion

Malgré les défis rencontrés, notamment sur les plateformes Cortex, Raspberry et QEMU, nous avons essayé de nombreuses choses avec des tentatives de code diverses et avons acquis une compréhension du développement d'une HAL avec les bases du code en Rust. Nos résultats sur Arduino et ESP32 nous offrent une base pour une éventuelle suite du projet.