Synchronisation temporelle de clefs ESP-32 avec le protocole Bluetooth

24//05/2022

Projet IoT

Hugo Bertin

Sommaire

Table des figures

1. Introduction

Un datacenter est un centre regroupant un nombre important de serveurs sur lesquelles des tâches variées sont réalisées. Certaines applications comme le travail autour de base de données, du réseau, ou encore des systèmes nécessitent une précision temporelle très importante. Cette dernière étant partagée entre différentes machines il est nécessaire de synchroniser les horloges des serveurs.

Le sujet de ce projet est d'utiliser des clefs utilisant le protocole Bluetooth pour implémenter un processus de synchronisation des horloges. Pour mettre en place cela, nous utilisons comme clefs, des microcontrôleurs ESP-32. Ce projet s'apparente à des produits vendus dans le commerce[1][2], échangeant des données temporelles via Bluetooth dans le but de se synchroniser des caméras, des enregistreurs audios ou encore des applications.

Nous voulons donc créer un système communiquant via un réseau sans fils, étant capable de se synchroniser sans l'intervention d'une référence temporelle extérieure. Cette synchronisation doit pouvoir être réalisée plusieurs fois. Effectivement les horloges se dérèglent avec le temps à cause d'imprécisions minimes cumulées.

Les ESP-32 ne disposant pas d'écran, ces derniers ont besoin d'être reliés à un périphérique extérieur (souvent un ordinateur) pour obtenir une console. Ils doivent donc être autonomes sur le long terme dans leur processus et ne pas nécessité d'intervention de l'extérieur.

Les ESP-32 étant des microcontrôleurs, leur capacité est limitée notamment au niveau de la mémoire, la programmation à réaliser doit donc être légère, orientée programmation embarquée.

2. Matériel

Comme mentionné ci-dessus nous utilisons deux microcontrôleur ESP32 développés par *Espressif System.* Le projet est implémenté dans le but d'être fonctionnel sur tous les types d'ESP32, cependant le développement et les tests ont seulement été réalisés sur des ESP32 ayant les références suivantes :

- ESP32-WROOM-32U
- ESP32-D0WDQ6.

Figure 1: ESP32-WROOM-32U

Un ordinateur est également nécessaire pour le développement ainsi que pour la visualisation de l'ensemble des processus. L'ESP32 n'intégrant pas de console, cette dernière peut être déportée sur un ordinateur. Un câble micro-USB est nécessaire pour relier le microcontrôleur à l'ordinateur.

3. Choix techniques

3.1 Environnement de développement

Les outils logiciels utilisés pour le développement de ce projet sont le Framework *ESP-IDF*[3] associé à l'éditeur de texte *Visual Studio Code*. Le Framework met à disposition les API de l'ESP, notamment l'API Bluetooth et de la documentation. Ce Framework peut être directement installé via une extension de *VSCode*, qui met à disposition des outils de configuration, de compilation, de téléversement et d'affichage console.

3.2 Langage de programmation

Les méthodes de l'API de ESP-IDF utilisent le langage de programmation C. Ce langage a donc été utilisé, il permet de rester à un bas niveau ce qui est adapté à la programmation sur un microcontrôleur pour utiliser de manière efficace les ressources limitées. Cela permet, par conséquent, une optimisation des performances.

3.3 Contrôle de version

Un repository git (consultable à cette adresse[4]) sur la forge GitHub a été mis en place pour ce projet.

4. Corps technique

4.1 Architecture du projet

Le projet est découpé en 2 partie différente, une pour chaque ESP-32. En effet, leurs comportements suivants des rôles différent l'un de l'autre, il est nécessaire que chacun ait son propre logiciel. L'architecture des fichiers du projet suit le diagramme ci-dessous.

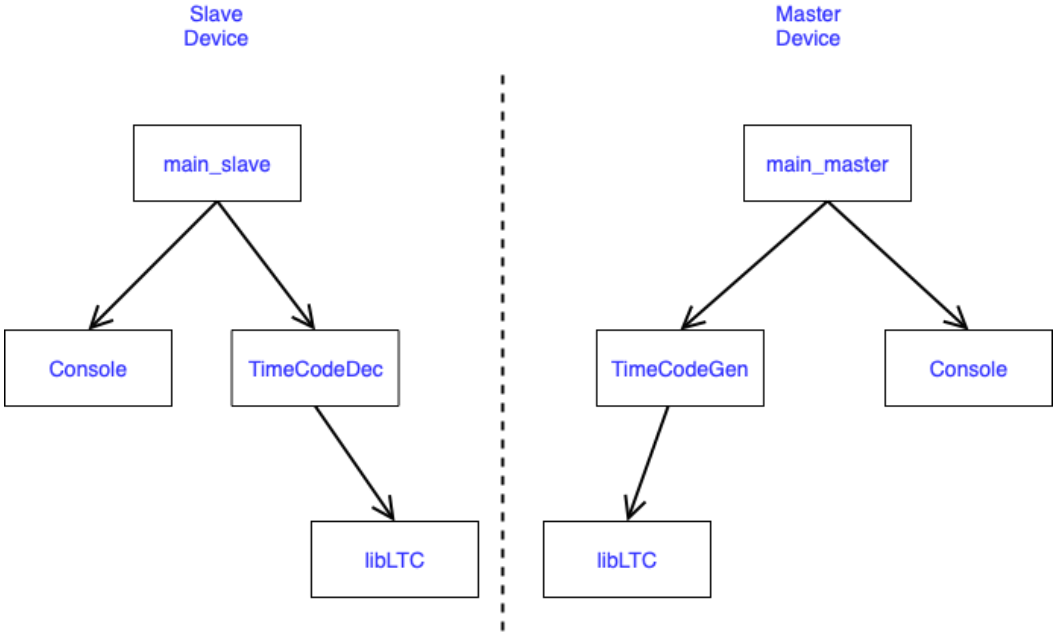

Figure 2: architecture des fichiers du projet

Pour chaque ESP 32, le fil principal est géré dans le fichier *main_name.c* qui fait appel aux méthodes de *console.c* qui gère l'interface en ligne de commande ainsi qu'aux méthodes de

timeCodeGen.c ou de *timeCodeDec.c* utilisant la librairie libLTC pour encoder ou décoder des références temporelles.

4.2 Connexion Bluetooth

Nous utilisons le protocole sans fil Bluetooth pour échanger les données entre les deux clés. Son implémentation est possible grâce à la stack Bluetooth intégré au sein des ESP32[5, p. 8]. Le framework de développement ESP-IDF nous met à disposition une API pour contrôler cette dernière.

Nous définissons deux rôles différents pour les esp32 :

- **Slave :** le périphérique attend une demande de connexion transmet ses informations lorsqu'un autre périphérique recherche les équipements disponibles.
- **Master :** recherche les autres équipements disponibles afin d'initier une connexion Bluetooth

La connexion entre les deux microcontrôleurs est réalisée en 2 étapes initiées par celui ayant le rôle de master :

1) La recherche des équipements Bluetooth visibles par le Master. Le retour contiendra les informations du Slave (adresse Bluetooth, nom…).
2) Le Master effectue ensuite une demande de connexion Bluetooth vers l'adresse du Slave récupérée. Le Slave accepte cette demande, les deux équipements sont donc connectés.

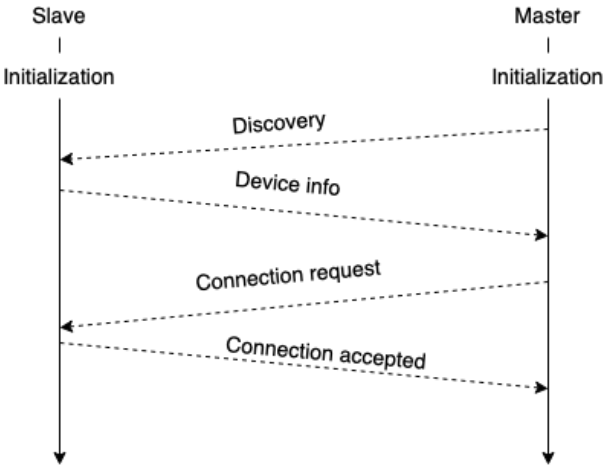

Figure 3: instanciation d'une connexion Bluetooth

Une fois la connexion instanciée, les ESP peuvent commencer à échanger des données dans les deux sens (Master vers Slave et inversement).

4.3 Echange de TimeCode

Les deux ESP ont besoin d'échanger des données temporelles via Bluetooth, cette transmission est réalisée à l'aide de timeCodes **SMPTE** sous forme d'un encodage **LTC (Linear Timecode).**

Les timecodes **SMPTE**[6] sont des représentations temporelles utilisées dans les différents domaines de la production audiovisuelle pour de la synchronisation audio ou vidéo. Elles se présentent sous la forme *hour:minute:second:frame*. Le champ *frame* correspond à la plus petite division temporelle (sous les secondes). Sa valeur est liée au débit images/secondes et correspond au pas entre deux images.

LTC (Linear TimeCode)[7] est un encodage de timecodes SMPTE utilisant un encodage de Manchester : un 0 logique est représentée par un front montant au centre du bit et un 1 logique par un front descendant. La figure 2 représente une trame.

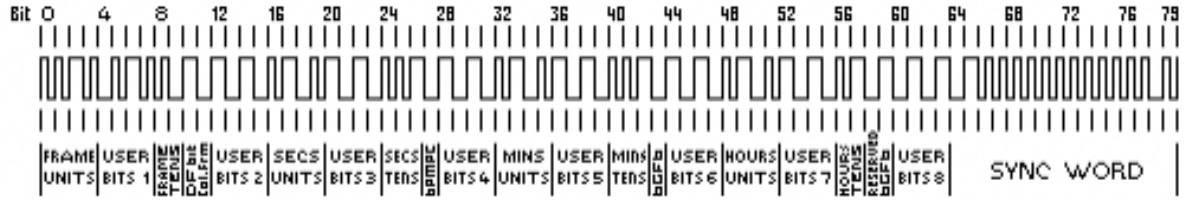

Figure 4: trame d'un LTC

Dans le projet, nous utilisons la librairie C open source **libLTC**[8] qui met à disposition des méthodes d'encodage et de décodage de timecodes LTC.

La synchronisation des horloges internes des deux ESP32 est réalisé en prenant comme référence l'horloge du premier ESP32. Le second ESP règle son horloge de manière à suivre cette référence. Les différentes étapes sont présentées sur schéma suivant.

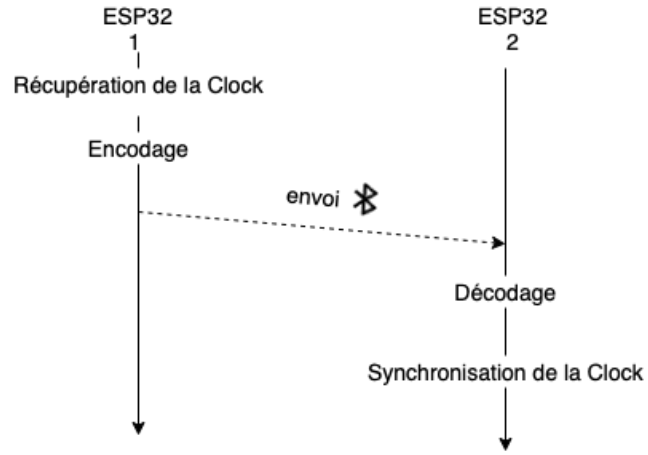

Figure 5: synchronisation des clocks des ESP32

1. Récupération de la valeur de l'horloge interne du premier ESP. Nous convertissons la valeur récupérée en millisecondes en TimeCode SMPTE. La valeur du champ *frame* prend la valeur arrondie la plus proche.
2. Encodage LTC : grâce à la bibliothèque libLTC
3. Transmission au second ESP32 via Bluetooth.
4. Réception du TimeCode encodé.
5. Décodage : avec libLTC
6. Mise à jour de l'horloge

4.4 Amélioration de la précision

La méthode de synchronisation présentée ci-dessus est fonctionnelle, cependant le délai entre la récupération de la valeur de l'horloge interne du premier ESP32 et la synchronisation de l'horloge du second est non négligeable. Effectivement les différentes étapes intermédiaires nécessitent du temps de calcul.

Les temps d'encodage et de décodage peuvent être facilement mesurés car ils se déroulent sur un seul et même ESP32, donc avec la même référence d'horloge. Il peut ainsi être pris en compte lors de la mise à jour de l'horloge du second ESP (cela implique la communication du temps d'encodage de l'ES32-1 vers l'ESP32-2).

Au contraire, la latence liée à la communication Bluetooth est partagée entre les deux microcontrôleurs. Leurs horloges n'étant pas (encore) synchronisés, une mesure ne faisant pas intervenir de référence temporelle extérieure n'est pas possible. Selon l'article[9] cette latence peut être exprimé selon la formule suivante :

$$Latency = Send\_Time + Access\_Time + Transmission\_Time + Propagation\_Time + Reception\_Time + Receive\_Time$$

Avec :

Send_Time : Temps d'assemblage du message et de demande de transmission,
Access_Time : Temps d'accès à un slot de transmission Bluetooth,
Transmission_Time : Temps nécessaire à l'émetteur pour envoyer l'intégralité du message,
Propagation_Time : Temps de propagation dans l'air, plus petit que la microseconde en Bluetooth donc négligeable,
Reception_Time : Temps pour recevoir l'intégralité du message, souvent similaire au *Transmission_Time*,
Receive_Time : Temps de traitement du message reçu pour que celui-ci soit disponible, similaire au *Access_Time.*

Afin d'estimer et compenser cette latence pour avoir une meilleure précision de synchronisation, une méthode est proposée. Cette méthode se base sur une version modifiée l'algorithme de Christian[10]. Voici une représentation de l'implémentation de la méthode pour notre projet.

1. L'ESP32-2 envoi une requête pour demander l'envoi de LTC, le temps T_send_request est récupéré.
2. L'ESP32-1 reçoit le message et récupère la durée de génération des timecodes durée_gen. Cette valeur est stockée.
3. L'ESP32-1 récupère sa valeur d'horloge T_m et procède à la génération du LTC correspondant.
4. L'ESP32-1 récupère le temps T_send_request puis envoie le LTC vers l'ESP32-2
5. L'ESP32-2 récupère le message et stocke le temps T_receive_ltc
6. L'ESP32-2 procède au décodage du LTC
7. L'ESP32-2 reçoit la valeur de la durée de génération de LTC.
8. L'ESP32 -2 peut procéder à la synchronisation de son horloge en suivant la formule suivante :

$$T_{new} = T_m + \frac{\sum_{i=1}^{N} \frac{\mathrm{T}_{\mathrm{receive}_{\mathrm{ltc}}} i - \mathrm{T}_{\mathrm{send}_{\mathrm{request}}} i - \mathrm{duree}_{\mathrm{gen}} i}{2}}{N} + duree_{gen} + duree\_dec$$

La formule ci-dessus utilise la moyenne de toutes les valeurs calculées pour la latence Bluetooth dans le but de réduire la probabilité d'erreurs liées à un écart entre les temps d'émission et réception. Cette moyenne des latences additionnée aux durées d'encodage et décodage constitue le décalage appliqué à la valeur d'horloge transmise par l'ESP32-1 pour synchroniser l'horloge de l'ESP32-2.

Ces différentes étapes sont représentées sur la figure 7 suivante.

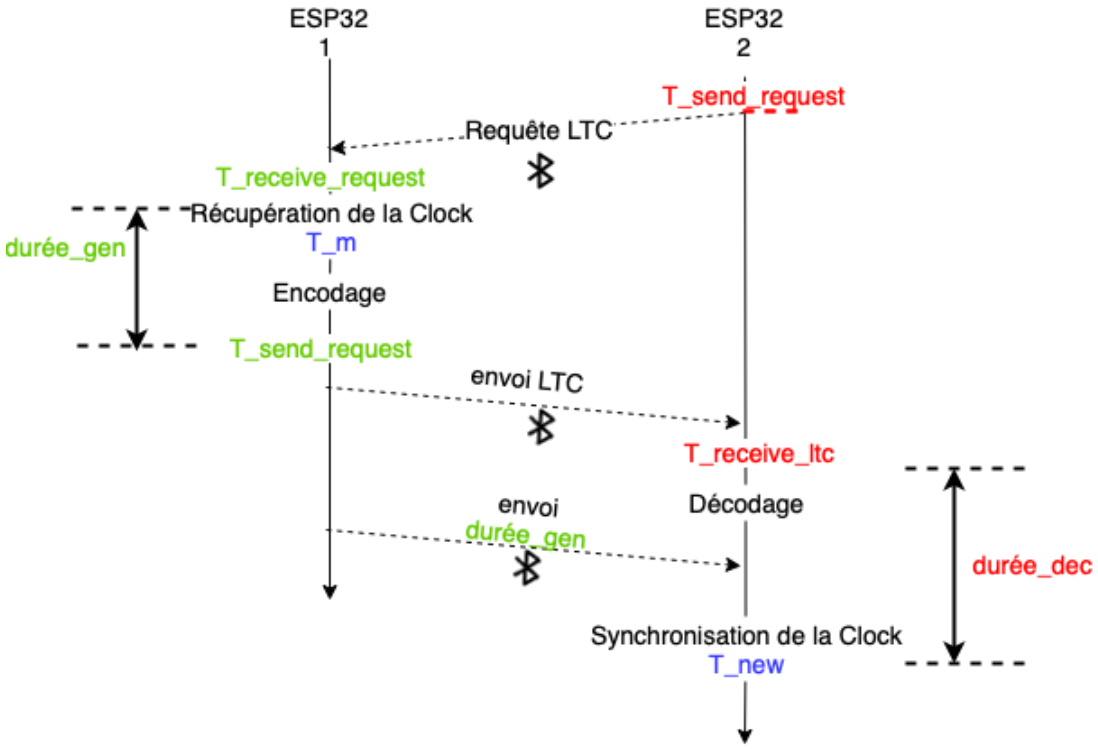

Figure 6: Méthode de synchronisation incluant les temps de calculs

5. Résultats et perspectives d'amélioration

La synchronisation des horloges des clefs ESP32 et la compensation de la latence Bluetooth semblent fonctionnelles. Les horloges paraissent synchronisées à l'œil nu lorsqu'elles sont affichées en console de l'ESP. Cependant, des mesures de synchronisation réelle n'ont pas été effectuées. En effet, il faut pour cela implémenter un système extérieur ayant une horloge de référence pour comparer les données transmises par les 2 ESP. Par exemple relier les ESP à un ordinateur et avoir une méthode pour récupérer exactement au même moment les valeurs de chaque horloge. Ce système aurait pour but de valider (ou rejeter) notre méthode de synchronisation.

Il y a également certaines limites au niveau de la précision, l'utilisation de TimeCode SMPTE limite cette dernière. En effet le délai minimum entre deux frames est de 33ms (pour une fréquence de 30 fps), ce qui est très supérieur au temps entre des tics d'horloge. Selon l'application voulu pour les serveurs derrière les ESP32, cette synchronisation peut être très insuffisante.

De plus des manques de stabilité persistent dans le système, en effet des erreurs peuvent être relevées au niveau des ESP-32 lors de la répétition des actions de synchronisation. Le temps a été insuffisant pour reproduire et résoudre ces erreurs.

6. Conclusion

Un processus de synchronisation d'horloge utilisant des timecodes LTC et communiquant par Bluetooth a été implémenté. Il intègre une compensation des temps de calculs et de transmission des informations pour augmenter la précision. Le système paraît fonctionnel mais

n’a pas été testé de manière rigoureuse. De nombreuses possibilités d’amélioration du projet sont possibles, la plus importante d’entre elles est la stratégie de validation.

Ce projet a été très enrichissant personnellement, il m’a permis de monter en compétences sur des sujets en rapport avec la formation IoT, notamment le réseau sans fil et la programmation embarquée. J’ai pu étendre mes connaissances sur les horloges, ce qui me servira probablement dans le futur.

7. Bibliographie

[1] ‘UltraSync BLUE | Timecode over Bluetooth’, *Timecode Systems*. https://www.timecodesystems.com/products-home/ultrasyncblue/ (accessed May 23, 2022).

[2] ‘Tentacle Original | tentacle sync’. https://tentaclesync.com/original (accessed May 23, 2022).

[3] ‘ESP-IDF Programming Guide - ESP32 - — ESP-IDF Programming Guide latest documentation’. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/ (accessed May 24, 2022).

[4] hbertin1, *hbertin1/esp32_sync_timecode*. 2022. Accessed: May 24, 2022. [Online]. Available: https://github.com/hbertin1/esp32_sync_timecode

[5] ‘esp32_datasheet_en.pdf’. Accessed: May 22, 2022. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[6] ‘SMPTE EBU timecode by Phil Rees’. Accessed: May 22, 2022. [Online]. Available: http://www.philrees.co.uk/articles/timecode.htm#smpte

[7] ‘Linear timecode’, *Wikipedia*. Feb. 18, 2021. Accessed: May 22, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Linear_timecode&oldid=1007571274

[8] ‘libltc: POSIX-C Library for handling Linear/Logitudinal Time Code (LTC)’. https://x42.github.io/libltc/ (accessed May 22, 2022).

[9] L. Lo Bello and O. Mirabella, ‘Clock synchronization issues in bluetooth-based industrial measurements’, in *2006 IEEE International Workshop on Factory Communication Systems*, Torino, Italy, 2006, pp. 193–202. doi: 10.1109/WFCS.2006.1704150.

[10] F. Cristian and F Cristian, ‘Probabilistic clock synchronization’, *Distrib. Comput.*, vol. 3, no. 3, pp. 146–158, 1989, doi: 10.1007/BF01784024.