Note de Rapport Android Filaire : Implémentation d’une interface filaire entre tablette Android et la Nucleo ST32F429

Auteur : Selso LIBERADO, [CIO Système Embarqués](https://www.ciose.fr/).

Avec le concours de Télécom Saint-Etienne.

Toute reproduction interdite sans autorisation de l’auteur.

|  |  |  |
| --- | --- | --- |
| Date | Auteur | Description |
| 05/04/2020 | Selso  LIBERADO | Création du document :   * mise en place du vcom, * mise en place des traces de debug |
| 24/05/2020 | Selso LIBERADO | Evolution :  Mise en place API de tet  Retour de test avec la tablette Android Samsung Galaxy Tab A |

Table des matières

[Rappel objectif 1](#_Toc37073114)

[Création du projet 1](#_Toc37073115)

[API utilisé pour la transmission 1](#_Toc37073116)

[Note pour aider au debug en ligne 2](#_Toc37073117)

[Le port COM associée au ST-Link 2](#_Toc37073118)

[Le port ITM 2](#_Toc37073119)

[Note d’Intégration de la couche dans le Protocole 5](#_Toc37073120)

# Introduction

Ce rapport présente la mise en œuvre par étape d’une communication filaire USB avec une tablette Android, du point de vue de la carte cible.

Plusieurs étapes sont définies :

* Prise en main des outils ST et de l’API pour créer un port COM et gérer le debug.
* Mise en place architecture d’intégration de code de test et du protocole
* Essai de communication avec des outils tierces
* Intégration protocole de communication avec l’IHM tablette et essais.

# Prise en main des outils ST et de l’API

## Rappel objectif

Mettre en œuvre une application minimale de port VCP (Virtual COM Port) sur le STM32 : réception et émission de caractères en utilisant un terminal sur PC.

Prendre en main les possibilités de debug.

Plutôt qu’un HelloWord envoyé par la carte l’écho permet de tester l’émission et la réception.

## Création du projet

On utilise STMCube pour générer le projet.

On active l’USB, avec sélection de la classe VCOM.

~~Aucune option ajoutée ou supprimée~~ , supprimer l’ethernet il fait planter le soft au démarrage (voir issue #1) 🡺 Peut-être à cause d’une HEAD insuffisante, mais peu importe.

On augmente la taille de la Heap à 0x600 apparemment la stack en fait usage.

Références :

Vidéo 48 minute de ST : [STM32 USB training - 09.1 USB CDC device basic labs](https://youtu.be/h9T0RTu9Muc)

## API utilisé pour la transmission

Les réceptions de données sont réalisées via l’appel de la callback déjà installée :

**static** int8\_t **CDC\_Receive\_FS**(uint8\_t\* Buf, uint32\_t \*Len)

Une zone de code « user » est prévue pour l’ajout du code. Dans notre exemple on a directement appelé la transmission.

Il faut s’attendre à ce que les transmissions des trames soient fractionnées : prévoir une machine à état pour la réception de trame complète, avec un mécanisme de resync.

Pour l’émission il faudra appeler la fonction suivante :

uint8\_t **CDC\_Transmit\_FS**(uint8\_t\* Buf, uint16\_t Len)

TODO :

* expliciter le retour de cette fonction, confirmer qu’elle est bloquante
* pour info : mode DMA possible.

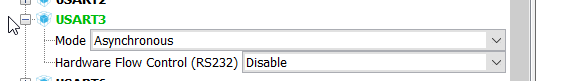
## Pour aider au debug en ligne

Outre le ST-Link des fonctionnalités de debug permettent de diagnostiquer sans arrêter le programme.

J’ai ajouté un fichier de traces qui permet de sélectionner une solution.

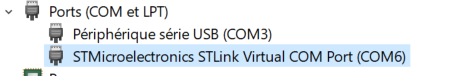
### Le port COM associée au ST-Link

Par défaut MX active l’USART3 sur la puce, celui-ci étant relié au ST-link qui fait le pont

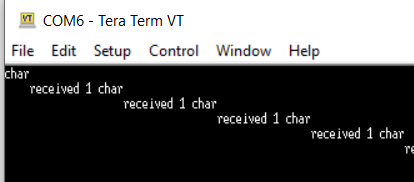


Les pins PA2 et PA3 sont disponibles sur la carte. Pour utiliser ce port il suffit d’utiliser la HAL pour envoyer des messages, sur la Nucléo c’est l’USART3.

Côté PC Le port COM du ST-Link est détecté :



Il suffit de l’ouvrir avec le terminal et d’observer les traces :



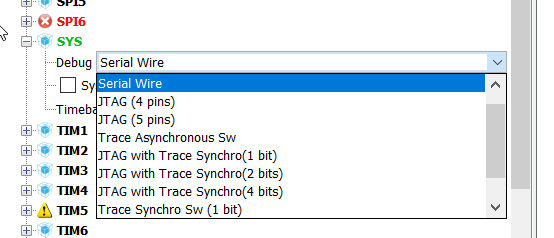
On notera qu’il faut configurer le terminal pour avoir le retour à la ligne.

### Le port ITM

L’avantage de ce fonctionnement est qu’il n’exige aucune connectique supplémentaire, est plus générique que le port COM du Stlink qui n’existe pas pour toutes les cartes. D’autre part le fonctionnement asynchrone ne pénalise pas l’exécution du code.

Il reste réservé à la phase de développement.

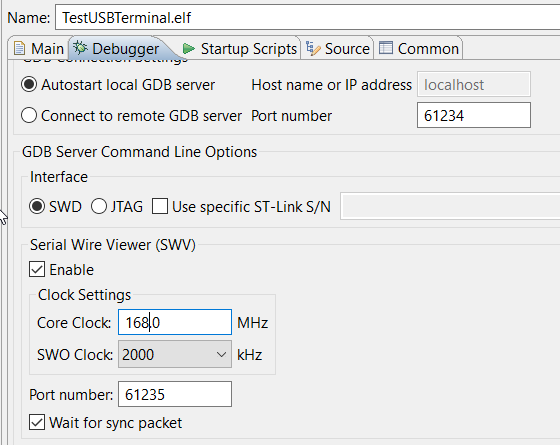
MX active le port  :



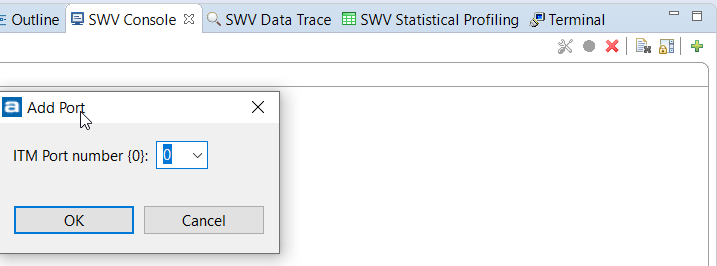
Il faut ajouter [un snippet de code](https://youtu.be/h9T0RTu9Muc?t=2587) pour envoyer les traces sur ce port. J’ai préféré le faire en utilisant cette [solution](http://blog.atollic.com/cortex-m-debugging-printf-redirection-to-a-debugger-console-using-swv/itm-part-1)[[1]](#footnote-1) de chez Atollic, plus élégante :

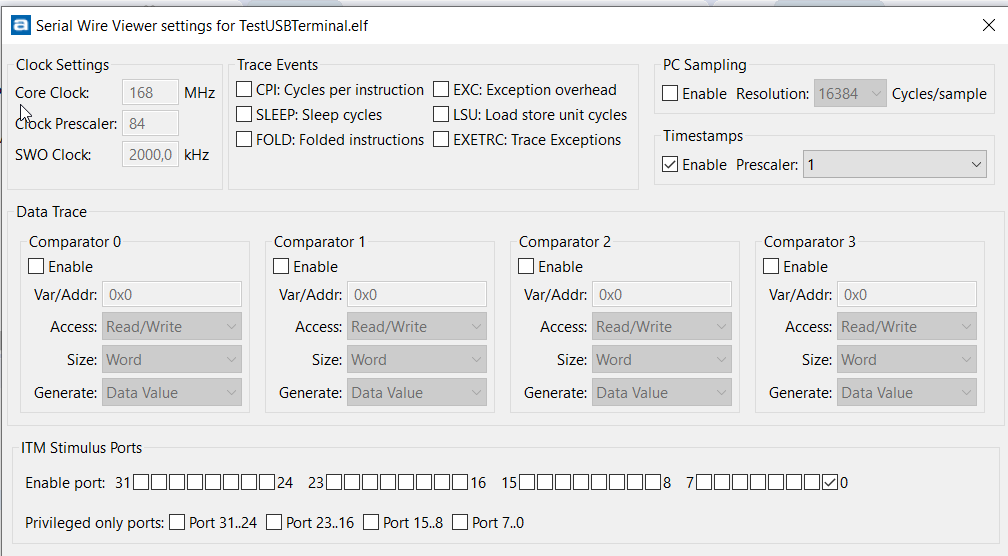
* J’ajoute un fichier trace qui permet de sélectionner la sortie de debug
* Je définis dedans la fonction d’écriture sur le port ITM0.

Côté PC, plutôt que d’utiliser le viel outil déprécié on peut utiliser Atollic :

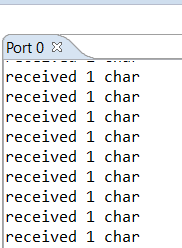


Après utiliser la vue dédiée pour activer la console. Il faut que l’exécution soit en pause pour configurer & activer cette console.





Ensuite cliquer sur start trace et relancer. Taper dans le terminal :



Note :

Beaucoup d’autres possibilité de traces sont possible, voir le [blog atollic](http://blog.atollic.com/cortex-m-debugging-printf-redirection-to-a-debugger-console-using-swv/itm-part-1).

Avec cette fonctionnalité on ne voit pas de port créé sous windows.

## Note d’Intégration de la couche dans le Protocole

L’implémentation dépendra de l’API.

L’implémentation doit permettre d’être portée sans modification sur pc : il faut créer une interface d’abstraction de la liaison.

# Architecture du code

## Architecture globale

L’architecture permet d’intégrer dans le même projet les tests unitaires précédents et l’intégration du protocole.

On se basera sur une interface qui listera des fonctions weak bien choisies qui couvriront nos besoins.

Les tests seront activés par des macros de compilation.

Tous les tests seront regroupés dans un répertoire « tst » :

Le fichier Tst\_iface.h liste les fonctions de l’interface :

/\*\* Boucle weak qui est appelée dans la boucle principale à la cadence d'1Hz

\*

\*/

**extern** \_\_weak **void** **tst\_loop\_1hz**(**void**);

/\*\* Fonction callback sur la réception USB

\* Pour la description des arguments voir CDC\_Receive\_FS

\* \*/

**extern** \_\_weak int8\_t **\_CDC\_Receive\_FS\_user**(uint8\_t\* Buf, uint32\_t \*Len);

Le fichier tst\_select.h permet sélectionner le ou les tests à activer en décommentant les symboles.

Pour l’illustration du mécanisme le code existant a été repris :

* Une macro active le test de l’écho sur terminal
* On a ajouté le test d’une fonction d’envoie continue « hello World »
* On a préparé une coquille pour l’intégration du protocole.

## Intégration du protocole

Pour l’intégration du protocole deux possibilités :

* On inclut les fichiers dans le projet directement
* On compile une bibliothèque statique dans un projet à part.

Pour l’interface avec l’USB soit :

* On ré-écrit une partie du protocole à partire de l’API LowLevel
* On créé un « device » approprié pour utiliser l’API slave :
* Il faudra surement placer dans une FIFO les réceptions USB (traitement asynchrone)
* Les écritures pour utiliser directement la fonction de transmission.

# Essai de communication avec des outils tierces

## 24/05/2020 Selso LIBERADO : test de l’écho & HelloWorld avec Android USBSerial Console

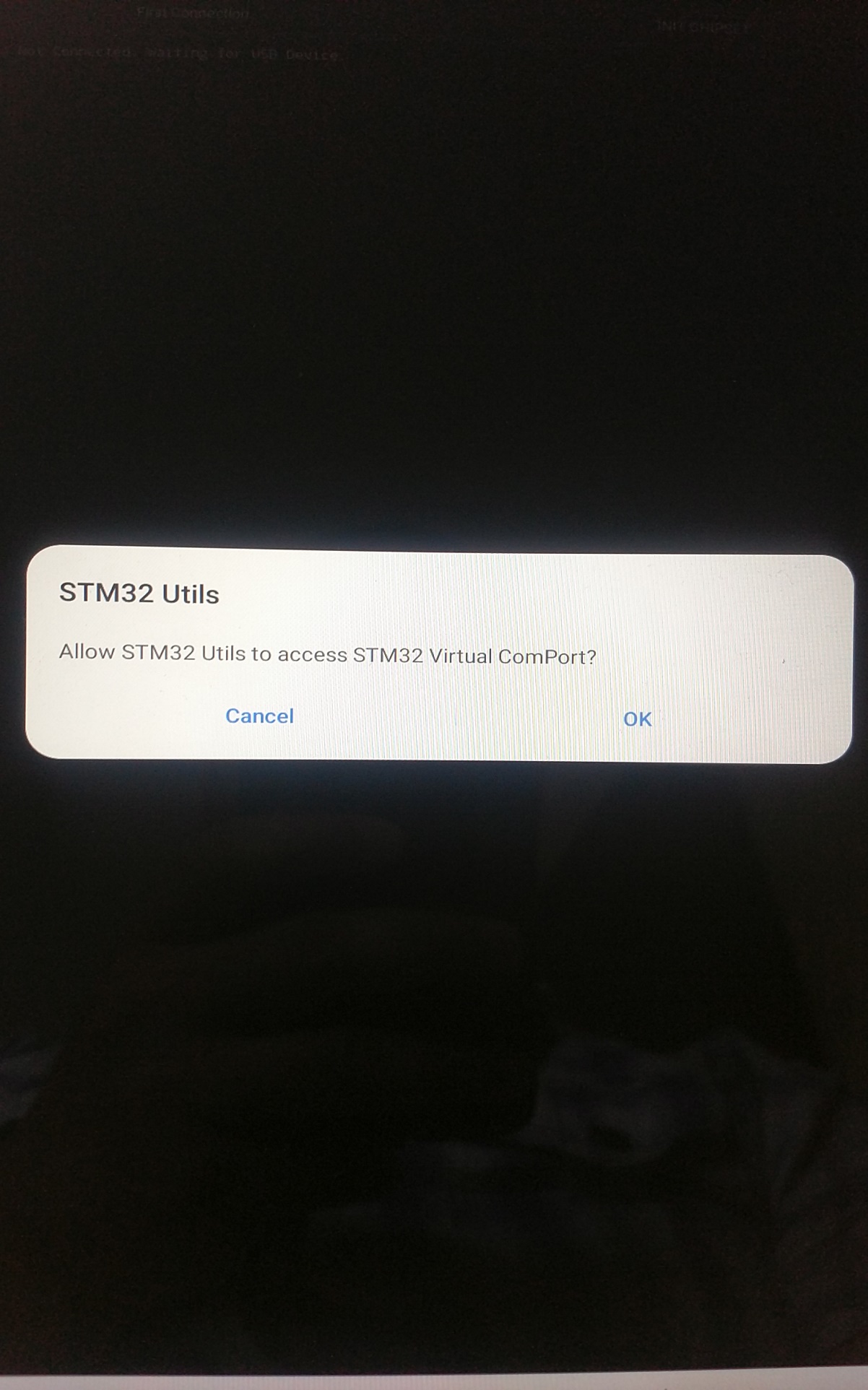
Installation de l’application [Android USBSerial Console](https://play.google.com/store/apps/details?id=jp.sugnakys.usbserialconsole&hl=fr) sur Android Samsung Galaxy Tab A.

Branchement de la tablette et de la cible comme suivant :

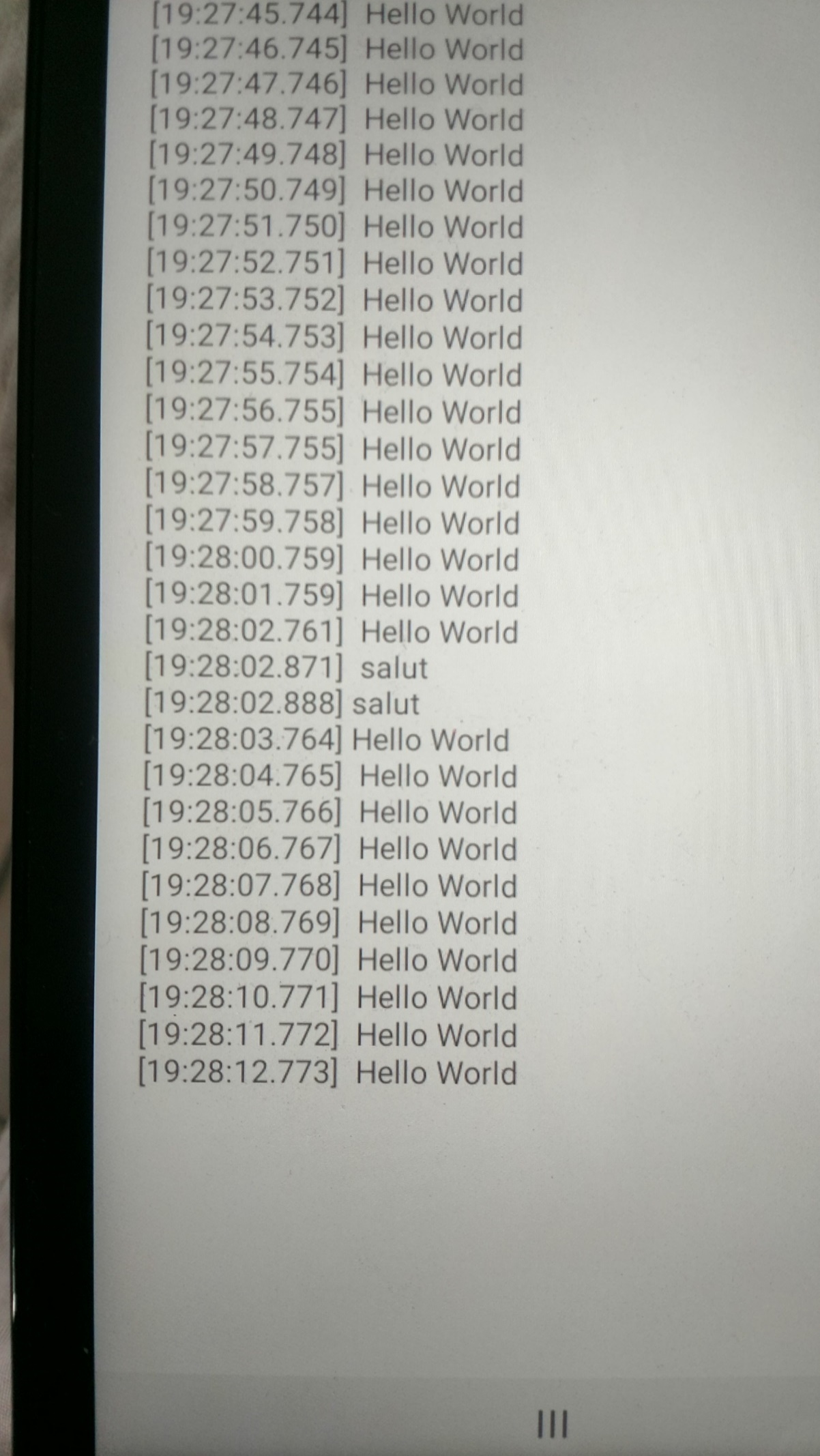


Un câble male uUSB type B vers femelle type A est utilisé avec un câble USB classique de connexion PC vers appareil.

Lorsque l’on lance l’application, celle-ci détecte la présence de la cible et demande l’autorisation d’y accéder :



Ensuite elle demande confirmation de connexion, ce après quoi l’application affiche alors les messages envoyés par la carte :



On notera que l’écho fonctionne très bien également (message « salut »).

Ce test démontre que la tablette supporte bien l’interface Vcom, et qu’une application est capable d’accéder à ce port.

En référence documentaire on liste le dépôt github de l’auteur du projet, qui peut être une piste pour l’intégration de notre protocole.

# Référence documentaire

TODO

API ST

Documentation de la carte.

Dépôts [github « UsbSerial »](https://github.com/felHR85) par Felipe Herranz.

1. Infos [complémentaires](https://www.openstm32.org/forumthread2893) par rapport à la carte. [↑](#footnote-ref-1)