Protocole TODO

# Introduction

C’est un protocole simple qui permet l’envoi de paquets entre des clients et des serveurs via des trames I²C. Il permet à l’expéditeur de s’authentifier, et de spécifier des options indiquant par exemple la présence ou non de cryptage.

## Motivations :

La technologie I²C ne permet pas à un destinataire de savoir quel composant I²C lui parle. Or une relation client-serveur impose la présence de réponse à certaines requêtes. Il a donc fallu créer une structure de communication : le paquet TODO.

## Principes :

* Le protocole TODO est une couche réseau se situant au-dessus de la couche liaison I²C.
* Ce protocole fonctionne en mode non connecté. En effet les communications I²C ne sont pas en mode connecté. Il n’est donc pas indispensable que le protocole TODO fonctionne en mode connecté. De plus, les cartes STM8S-Discovery ne sont pas assez puissantes pour supporter un protocole connecté.
* L’en-tête du paquet TODO est modulable grâce au bit MO (voir Entête du paquet TODO). Cela permet l’évolution future du protocole en assurant la rétrocompatibilité.

# Configuration et utilisation

## Configuration :

Le protocole TODO s’appuie sur la technologie I²C.

* Adresse (unique sur le réseau)
* Adresse sur 7bits
* Un bit de direction configuré en écriture
* Fréquence
* Rapport cyclique
* Type d’acquittement
* InputClockFrequencyMHz

## Utilisation :

Le protocole doit laisser au développeur la liberté de communiquer en I²C sans utiliser des paquets TODO.

Il y a donc possibilité de désactiver ce dernier. Il faut pour cela appeler la fonction TODO\_freeze(u8). Si le protocole est freezé, il appellera une autre fonction lors d’une interruption I²C. C’est au développeur de définir cette fonction.

Conseil : Il est fortement conseillé d’être le maître I²C avant de freeze.

# En-tête du paquet TODO

Les données échangées posséderont l’en-tête suivant.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit7** | **Bit6** | **Bit5** | **Bit4** | **Bit3** | **Bit2** | **Bit1** | **Bit0** |
| Adresse source (expéditeur) | | | | | | | 1 |
| TY | | CR |  |  |  |  | MO |
|  |  |  |  |  |  |  | MO |
| Données | | | | | | | |
|  |  |  |  |  |  |  |  |

* TY : définie le type de l’entité :
  + 00 : Terminal
  + 01 : Serveur
  + 10 : Périphérique
  + 11 : Autre
* CR : Utilisation du cryptage :
  + 0 : non
  + 1 : oui
* MO : More Option :
  + 0 : Les octets suivants sont des données
  + 1 : Il y a au moins un octet supplémentaire d’option.
* Données : Données contenues par la trame, terminées par un octet 0x00.

# Gestion des erreurs :

* Si une entité A s’adresse à une autre entité B en mode lecture alors B flood A jusqu’à la fin de la communication. (interrupt ADDRESSSENTMATCHED).
* Si les données ne se terminent pas par la caractère nul, alors le paquet est ignoré.

# Interface pour l’applicatif :

* init(Type, uint8 addr, uint CRYPT\_KEY) :
* aux\_interrupt( (voi\*)(Func\*)(void) ) :
* recv() : (interrupt RXNOTEMPTY ). bloquant, remplie un buffer statique
* send(uint8 addr, char\* data, uint16 size) :
* freeze(uint8) : unfreeze(0), freeze( !0 )
* close() :

# Ouverture :

## Gestion dynamique des adresses.

* Paquet un bit NO\_ADDRESS demandera une adresse serveur.

Commande déjà prete ?