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.

## Pourquoi créer ce protocole ?

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. Ainsi, l’adresse TODO permettant d’identifier une entité d’une autre est la même que l’adresse I²C. Cette adresse est donc sur 7 bits et doit être unique. De plus, elle doit respecter les classes d’adresses spéciales ou réservées de la technologie I²C telle que l’adresse de broadcast.

Dans ce protocole, une entité à laquelle on se connecte en lecture n’a pas de sens. Le 8ème bit, servant à configurer le mode lecture ou écriture en I²C sera donc tout le temps configuré en mode écriture. Cependant en I²C, l’esclave n’a pas le contrôle de la transmission. Si on se connecte en lecture, il faudra donc que l’esclave réponde au maître jusqu’à ce que ce dernier termine la transmission. Le protocole TODO impose donc à l’esclave de répondre au serveur, mais n’est pas regardant vis-à-vis du contenu de la réponse.

L’acquittement d’un octet concernera l’octet courant.

## Implémentation sur un STM8S :

Nous implémenterons le protocole à la vitesse de communication standard de l’I²C, 100kHz. Premièrement, il faut mettre le bit F/S de I2C\_CCRH à 0. Nous voulons ensuite une période de 10µs. La période de Signal CLock (SCL) vaut, en mode standard, 2\*CCR\*.

Ainsi pour un registre FREQR de 16MHz (FREQR = 0x10) nous aurons . Il faut donc placer Clock Control Register (CCR) à 80 (soit 0x50) pour avoir .

A la vitesse de communication standard, le rapport cyclique n’a pas d’impact. Le bit DUTY\_CYCLE du I2C\_CCRH sera donc arbitrairement mis à 0.

Comme indiqué précédemment, l’acquittement se fait sur l’octet courant. Ainsi le bit ACK de I2C\_CR2 sera positionné tandis que le bit POS de I2C\_CR2 sera à 0.

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 :

# Interface pour l’applicatif :

* init(Type, uint8 addr, uint CRYPT\_KEY) :
* aux\_interrupt( (void\*)(Func\*)(void) ) :
* int send(int s, const void \*msg, size\_t len, int flags);
* int recv(int s, void \*buf, int len, unsigned int flags);
* freeze(uint8) : unfreeze(0), freeze( !0 )
* close() :

# Ouverture :

## Gestion dynamique des adresses.

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

Commande déjà prete ?