

# Pré-spécification du SHER-Bus

## Avertissement

Le contenu n'est pas fixe et peut être modifié sans préavis !

SHER-Bus Stand for:

Systemwide Hub for Efficient Routing Bus

and

SHER-Bus Handles Extensive Resource Bridging, Unifying Systems



## Qu'est-ce que SHER-Bus ?

Il s'agit d'une conception de bus série permettant de regrouper de nombreux bus série à basse vitesse trouvés dans l'électronique moderne en une ou plusieurs paires de différentiels à grande vitesse. Des exemples de bus série basse vitesse sont SPI, I2C, CAN, PWM, PCM, etc. Fini le jour où vous réalisez que vous n'avez pas assez de bus i2c ou UARTS.

Ce bus peut également prendre en charge des usages plus généraux comme l'interface homme-machine (clavier, clavier, manette de jeu, joysticks), l'audio (I2S, SPDIF), la vidéo basse résolution (oleds monochromes, spi tft lcd), le contrôle moteur (pas à pas) , Entrée-sortie à usage général (GPIO), gestion de la batterie (SOC, DOD, SOH, série, parallèle, courant de charge/décharge, tension, températures), etc.

SHER-Bus est également idéal pour la communication à usage général (comme UART, JASON, protobuf, modbus, etc.) entre les contrôleurs.

## Pourquoi SHER-Bus ?

Sherbus est né de la frustration de travailler avec les nombreux protocoles de communication volumineux, lents ou propriétaires des appareils embarqués. Sherbus est léger et facile à comprendre par quiconque, du fabricant au designer chevronné. Le protocole est modulaire (Seule la couche Protocole[P]est obligatoire). Cette modularité donne aux responsables de la mise en œuvre la possibilité d'adapter la communication au produit et non au contraire. Les développeurs n'auront pas besoin de gérer une pile complexe pour quelque chose de simple (comme par exemple une classe USB CDC complète pour un UART de base). La complexité du logiciel est conçue pour évoluer avec la complexité du bus. En d'autres termes, les tâches faciles sont faciles à réaliser sur un simple bus. Sinon, plus de fonctionnalités sont ajoutées, plus il faut faire attention (par exemple, identification de la position du bus, voies multiples, etc.) pour garantir l'intégrité du bus. Cela signifiait que la barre d'entrée était très basse, mais que le bus restait suffisamment puissant pour gérer des tâches complexes lorsque les développeurs en avaient besoin. C'est l'innovation SHER-Bus, car les protocoles actuels sont soit trop complexes pour une tâche simple (une pile USB peut occuper la majeure partie de l'espace programme sur la plupart des microcontrôleurs, même Tiny-USB), soit trop simples pour

une tâche complexe. SHER-Bus est à la fois simple pour une tâche simple et plein de ressources pour l'utilisateur expérimenté.

Une autre raison pour laquelle nous avons besoin d'un protocole comme SHER-Bus est que, même si RISC-V a contribué à ouvrir le marché des processeurs à une IP sans licence pour les processeurs, de nombreux systèmes sur puce nécessitent l'utilisation d'une IP propriétaire pour sa connectivité. L'un des objectifs de SHER-Bus est qu'un jour un SOC 100 % gratuit et open source arrive sur le marché. Avec l'aide de SHER-Bus et de nombreux autres projets comme celui-ci, la communauté open source peut atteindre cet objectif.



## Conception d'autobus

**Avertissement** M LVDS n'est pas un choix fixe et est sujet à des tests et à une comparaison de prix (les émetteurs-récepteurs LVDS sont chers !)

Le bus est basé sur le M-LVDS (alias TIA/EIA-899). Il est conçu pour prendre en charge le multipoint dès le départ. Le bus est filaire-ou (niveau haut dominant). Jusqu'à 32 appareils (30 contrôleurs/pont et 2 ponts END) peuvent être connectés à une seule voie différentielle. 30 appareils peuvent sembler limités mais la limite théorique des appareils i2c sur SHER-Bus est de 22098!^[1]Plusieurs connexions série (comme dans un fond de panier) peuvent être ajoutées pour augmenter le débit. L'horloge est intégrée au flux de données, il n'est donc pas nécessaire d'ajouter une piste d'horloge. Une horloge optionnelle peut être ajoutée pour synchroniser les fonctions telles que l'audio. Il utilise un codage de 8 bits à 10 bits (ou Manchester idk encore) sur la couche physique pour garantir l'équilibre DC et fournir une première couche de vérification des erreurs.

[^1] : 127 appareil i2c\_6 ports maîtres i2c sur 1 pont\_29 ponts (il nous faut au minimum 1 contrôleur) = 22098

## Appareils de bus



Il n'existe que 3 types d'appareils qui remplissent des fonctions différentes dans le réseau de bus.

First there is the controller that generate data packet for other to parse. They give the bus his function. for instance, the controler can give command to a bridge for him to read an i2c temperature sensor. The controller then interpret that data and then adjusting an spi DAC to output a value for the control of a Fan. They can also give command to other controller on the same bus. So in other word their job is to be the bain of the communication. They consist of mostly of Microcontroller, Embedded computer(Raspberry pi) or FPGA.



En deuxième lieu, il y a le pont dont la tâche consiste à combler le fossé entre le nouveau bus et le protocole déjà existant. Ils peuvent avoir plusieurs bus série (jusqu'à 6 et 1 canal de contrôle général utilisant[IPB]). Ils sont principalement constitués d'ASIC ou de FPGA. Bien qu'à l'époque ou en écrivant aucun code ASIC ou FPGA n'ait été fabriqué/écrit. Le microcontrôleur peut également faire office de pont. Brige peut être intégré dans une conception déjà existante sous forme de matrice ou dans un boîtier multi-matrice. En réutilisant plusieurs fois la conception, cela contribue à réduire la complexité de la refonte et à accélérer la mise sur le

marché. Les responsables de la mise en œuvre du pont doivent faire attention aux licences des bus existants, car certains ne sont pas libres de les mettre en œuvre.



En troisième lieu, il y a le pont d'extrémité dont le travail consiste à connecter plusieurs bus SHER ensemble ou d'autres bus HAUTE VITESSE (comme USB, SDIO ou Ethernet). Ils terminent le bus (là où se trouve la résistance de terminaison), c'est pourquoi ils sont limités à 2 qui peuvent être connectés sur le bus. Ils sont complexes et non obligatoires.

## Architecture des paquets

---

L'architecture des paquets suit la même philosophie que celle de l'architecture du processeur RISC-V. Un seul jeu d'instructions est obligatoire (P pour SHER-Bus) et chaque jeu est étiqueté par une lettre. (ex. : I M C A F D Q pour CPU risc-V) ou un mot/achronim (ex. : Zicsr). Ce qui est différent en raison de la nature de la communication, c'est que chaque couche est intégrée dans l'ensemble des couches situées en dessous. Par exemple, un paquet audio (A) est intégré dans une identification de protocole de bus.[IPB]qui est à son tour intégré dans un package de flux (S) qui est à son tour intégré dans un package de protocole (S). Nous avons une structure comme P(S(BPI(A))), mais si ce message audio est adressé à tout le monde dans le bus, nous pouvons supprimer la couche BPI et avoir un message structuré comme ceci : P(S(A)). Cela augmente considérablement la flexibilité du bus. Un récepteur peut effectuer un masque ET sur l'ensemble du message pour voir si le message l'intéresse et éliminer ceux qui ne le sont pas (ex. : un pont I2C ne se soucie pas d'un package audio (A) mais un pont I2S le fait. ).



## Couche de protocole (P)

La couche protocole est la seule couche obligatoire de la spécification. Il gère le minimum pour être une transaction valide. L'implémenteur utilise cette couche pour envoyer un message de très bas niveau comme une communication point à point de type UART, puisque l'adressage est géré à un niveau supérieur, seule une configuration de bus point à point ou multipoint est possible en utilisant uniquement ce niveau. Il s'agit d'une fonctionnalité car dans de nombreuses applications, vous ne voudriez pas d'une pile lourde pour quelque chose de simple. Cela accorde également à l'implémenteur la liberté de créer une pile de protocoles personnalisée pour les applications qui ne sont pas couvertes par la pile existante. (ex. : SAE J1939 et CanOPEN sont tous deux des piles construites sur la nature non restrictive du protocole CAN, SHER-Bus essayant de le faire. le même mais gratuit avec une pile commune qui aide l'implémenteur de pont et l'implémenteur de bus à avoir un terrain d'entente avec lequel travailler). le premier octet sert à indiquer s'il s'agit d'un paquet standard ou personnalisé. Un un (1) sur le MSB du premier octet indique que la charge utile est un message conforme au bus SER. L'implémenteur peut envoyer un message personnalisé en définissant le premier octet sur 0 (0x00).



## Couche réseau



## Control Messages (C)

Les messages de contrôle sont utilisés pour modifier la façon dont le bus ou un appareil réagit. Par exemple, un implémenteur peut effectuer une réinitialisation de l'appareil. Il est également utilisé par le [IPB]layer for Dynamic Addressing.

### **Messages d'interruption (I)**

Les messages d'interruption sont conçus pour être aussi proches que possible d'une interruption informatique. Ils peuvent utiliser le [IPB]mais sont davantage destinés à l'attention de Bus Wise. Par exemple, un ADC peut envoyer une interruption pour indiquer que de nouvelles données sont prêtes à être lues.

### **Messages Boomerang (B)**

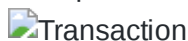
Les messages Boomerang, comme leur nom l'indique, sont destinés à revenir à l'expéditeur. Dès réception d'un message (B), le récepteur l'utilise pour effectuer une action, puis renvoie le même message avec une modification en fonction de cette action. Les messages (B) constituent une exception car ils doivent prendre en charge les messages d'identification de position du bus, sinon tout le monde dans le bus renverrait le message. Le renvoi du même message garantit également qu'il a été reçu correctement et permet la désynchronisation de la transaction (les allers-retours peuvent se produire avec d'autres messages les séparant). Un exemple serait une lecture I2C sur le bus. Un contrôleur peut demander la lecture de l'adresse W adresse X i2c sur le bus Y I2C du pont Z. Le pont répondrait après avoir effectué la lecture qu'il avait reçu n octets de données de l'adresse W X i2c valide sur le bus Y I2C en renvoyant le même message mais en échangeant l'octet récepteur/transceiver du [IPB]et ajouter les données lues.

### **Flux de messages (S)**

Les messages de flux sont destinés lorsque l'intégrité des données n'est pas importante mais qu'un flux constant de données l'est. Les messages de flux sont destinés à un flux audio par exemple. Ce n'est pas mal si 1 ou 2 paquets sont perdus, il vaut mieux avoir un flux constant de paquets. Les messages de flux sont meilleurs s'ils se trouvent sur un bus séparé, car ils ont la priorité la plus basse.

### **Identification de la position du bus (BPI)**

L'identification de la position du bus donne au contrôleur un moyen de parler avec un appareil spécifique tandis que les autres restent inchangés. Bridge doit prendre en charge BPI car ils ne savent jamais de quel bus il fera partie. Chaque appareil peut avoir jusqu'à 7 sous-appareils. Le sous-appareil 0 est réservé au contrôle ou lorsque les sous-appareils ne sont pas utilisés. L'appareil peut obtenir une adresse de 3 manières possibles : statiquement, pseudo-dynamiquement et dynamiquement.



### **Adressage statique**

Chaque appareil sur le bus a une adresse prédéfinie qui ne change pas. Il est de la responsabilité de l'implémenteur du bus de définir une adresse unique pour chaque appareil présent sur le bus. Ce mode d'adressage est utile lorsque l'implémenteur connaît la disposition du bus et qu'elle ne change pas comme dans un système entièrement fermé sans ports extérieurs. A documenter.

### **Pseudo-Dynamic Addressing**

Chaque appareil obtient son adresse par un moyen externe. Par exemple, dans un fond de panier, chaque emplacement est étiqueté électroniquement (gpio ou eeprom i2c) avec une adresse unique. Le périphérique

SHER-Bus lit cette adresse lors de la mise sous tension et l'utilise pour la communication. A documenter.

**Dynamic Addressing**


Chaque appareil obtient son adresse en demandant au SHER-Bus à l'aide du message Control(C). A documenter.

**Messages de haut niveau**

Avertissement

A écrire

Exemple de transaction

Protocol stack