Pré-spécification du SHER-Bus

> [!IMPORTANT]
>
> pour des questions et commentaires utiliser la section
> 
> . Pour plus de détail sur la structure du PMC allez
> 
> .

> [!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 ?

SHER-Bus est 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. Les exemples de bus série basse vitesse incluent SPI, I2C, CAN, PWM, PCM, etc. L'époque où vous réalisez que vous n'avez pas assez de bus i2c ou UARTS est révolue.

Ce bus peut également prendre en charge un usage plus général comme l'interface homme-machine (clavier, clavier, contrôleur 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, JSON, protobuf, modbus, etc.) entre les contrôleurs.

Pourquoi SHER-Bus ?

Sherbus est né de la frustration liée au travail avec les nombreux protocoles de communication volumineux, lents ou propriétaires des appareils embarqués. Sherbus est léger et facile à comprendre par tous, des créateurs aux designers chevronnés. Le protocole est modulaire (Seule la couche Protocole[P]est obligatoire). Cette modularité donne aux responsables de la mise en œuvre une marge de manœuvre pour adapter la communication au produit et non l'inverse. 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 peuvent être facilement réalisées sur un simple bus. Sinon, plus de fonctionnalités sont ajoutées (par exemple, identification de la position du bus, voies multiples, etc.), plus il faut veiller à garantir l'intégrité du bus. Cela signifie que la barrière à l'entrée est très faible, mais que le bus reste suffisamment puissant pour gérer des tâches complexes lorsque les développeurs en ont besoin. C'est l'innovation de SHER-Bus, car les protocoles actuels sont soit trop complexes pour des tâches simples (une pile USB peut occuper la majeure partie de l'espace programme sur la plupart des microcontrôleurs, même Tiny-USB), soit trop simples pour des tâches complexes. SHER-Bus est à la fois simple pour les tâches simples et extensible pour les utilisateurs expérimentés.

Une autre raison pour laquelle un protocole tel que SHER-Bus est nécessaire est que, même si RISC-V a contribué à ouvrir le marché des processeurs à une IP sans licence pour les processeurs, de nombreuses conceptions de systèmes sur puce (SoC) nécessitent l'utilisation de protocoles propriétaires. IP pour leur 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.

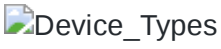
Bus design

> **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 !)

The Bus is based on the M-LVDS(aka, TIA/EIA-899). It's design to support multipoint from the get go. The bus is wired-or (level high dominant). Up to 32 devices (30 controlers/bridge and 2 END bridges) can be connected to a single differential lane. 30 devices can sound limited but the theorical limit of i2c device on SHER-Bus is 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 en option peut être ajoutée pour synchroniser des fonctions telles que l'audio. Il utilise un codage de 8 bits à 10 bits (ou Manchester idk pour l'instant) sur la couche physique pour garantir l'équilibre DC et fournir une première couche de vérification des erreurs.

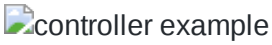
[^1] : 127 appareil i2c_6 ports maîtres i2c sur 1 pont_29 ponts (il nous faut au minimum 1 contrôleur) = 22098

Appareils de bus

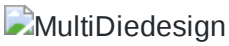


Il n'existe que 3 types d'appareils dans le réseau de bus, chacun remplissant une fonction différente.

Il y a d'abord le contrôleur qui génère des paquets de données que d'autres peuvent analyser. Ils donnent au bus sa fonction. Par exemple, le contrôleur peut donner des commandes à un pont pour qu'il lise un capteur de température i2c. Le contrôleur interprète ensuite ces données et ajuste un spi DAC pour émettre une valeur pour le contrôle d'un ventilateur. Ils peuvent également donner des commandes à d'autres contrôleurs sur le même bus. En d'autres termes, leur travail consiste à être le cerveau de la communication. Ils sont constitués principalement de microcontrôleurs, d'ordinateurs embarqués (Raspberry pi) ou de FPGA.



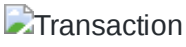
Deuxièmement, il y a le pont dont la tâche est de combler le fossé entre le nouveau bus et les protocoles déjà existants. Ils peuvent avoir une multitude de bus série (jusqu'à 6 et 1 canal de contrôle général utilisant[IPB]). Ils sont principalement constitués d'ASIC ou de FPGA. Bien qu'au moment d'écrire ces lignes, aucun code ASIC ou FPGA n'ait été fabriqué/écrit. Les microcontrôleurs peuvent également servir de ponts. Les ponts peuvent être intégrés dans des conceptions déjà existantes en matrice ou en boîtier multi-matrice. Conformément à la règle de la conception une fois, réutilisation partout, cela permet de réduire la complexité de la refonte et d'accélérer la mise sur le marché. Les responsables de la mise en œuvre des ponts doivent être prudents en ce qui concerne les licences des bus existants, car certains ne sont pas libres de les mettre en œuvre.



Troisièmement, 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 adoptée par 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 à la charge utile 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 à son tour est intégré dans un package de protocole (P). On a donc une structure comme P(S(BPI(A))), mais si ce message audio est adressé à tout le monde dans le bus on peut supprimer la couche BPI et avoir le 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 d'un I2S. passage du pont).

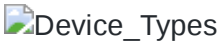


Couche de protocole (P)

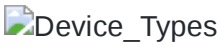
La couche protocole est la seule couche obligatoire de la spécification. Il gère la charge utile minimale pour être une transaction valide. Il se compose d'une seule impulsion d'horloge suivie de 32 octets codés en 8b/10b. Les développeurs utilisent cette couche pour envoyer des messages de très bas niveau, comme une communication point à point de type UART.[^2], puisque l'adressage est géré à un niveau supérieur, seules les configurations de bus point à point ou multipoint sont possibles en utilisant uniquement ce niveau. Il s'agit d'une fonctionnalité car dans de nombreuses applications, vous ne voudriez pas une pile lourde pour quelque chose de simple. Cela donne également aux développeurs la liberté de créer des piles de protocoles personnalisées pour les applications qui ne sont pas couvertes par les piles existantes. (Ex. : SAE J1939 et CanOPEN sont deux piles construites sur la nature non restrictive du protocole CAN, SHER-Bus essaie de faire la même chose mais gratuitement en ayant une pile commune afin d'aider les implémenteurs de pont et les implémenteurs de bus commencent du même point commun). Le premier octet indique 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. Les développeurs peuvent envoyer des messages personnalisés en définissant le premier octet sur 0(0x00).

[^2]: seuls 2 contrôleurs doivent être connectés sur le bus. Un UART multipoint doit être disponible dans la couche application.



Couche réseau



Control Messages (C)

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

Messages d'interruption (I)

Les messages d'interruption sont conçus pour être aussi proches que possible des interruptions informatiques traditionnelles. Ils peuvent utiliser le[IPB] but are more meant for Bus Wise attention. For example An ADC can send a Interupt to say that fresh data is ready to be read.

Messages Boomerang (B)

Les messages Bomerang, comme leur nom l'indique, sont destinés à revenir à l'expéditeur. Dès réception d'un message (B), le destinataire 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. Renvoyer le même message garantit également qu'il a été reçu correctement et permet une communication asynchrone - c'est-à-dire une synchronisation par transaction (les allers-retours peuvent se produire avec d'autres messages les séparant). Un exemple serait une lecture I2C sur le bus. Un contrôleur pourrait demander la lecture de l'adresse W X esclave i2c sur le bus Y I2C du pont Z. Le pont répondrait après avoir effectué la lecture en recevant n octets de données de l'esclave i2c de l'adresse W sur le bus Y I2C en renvoyant le même message mais en échangeant l'octet récepteur/émetteur-récepteur du[IPB]et en ajoutant les données lues.

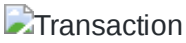
Flux de messages (S)

Les flux envoyés par message sont destinés au cas d'utilisation dans lequel l'intégrité des données n'est pas importante mais un flux constant de données l'est. Les messages de flux sont destinés à un flux audio par exemple. Ce n'est pas grave si 1 ou 2 paquets sont perdus car 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 aux contrôleurs un moyen de communiquer avec un appareil spécifique tandis que les autres restent inchangés. Les ponts doivent prendre en charge BPI car ils ne savent

jamais de quel bus ils feront 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. Les appareils peuvent obtenir des adresses de trois manières : statiquement, pseudo-dynamiquement et dynamiquement.



Adressage statique

Chaque appareil sur le bus possède 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 en utilisant un mécanisme externe. Par exemple, dans un fond de panier, chaque emplacement est étiqueté électroniquement (gpio ou eeprom i2c) avec une adresse unique. Les appareils SHER-Bus lisent cette adresse lors de la mise sous tension et l'utilisent pour la communication. A documenter.

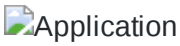
Dynamic Addressing

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

Messages de haut niveau

> **Avertissement**A écrire

Les messages de haut niveau sont des fonctions que le bus peut assumer. Du protocole de transport hérité à une communication basée sur des fonctions (gestion de la batterie, enregistrement des données, vidéo, audio), SHER-Bus peut le faire. Chaque application est définie par un code d'application qui est donné par la communauté BUS. (À DÉTERMINER)



Exemple de transaction

Dans ces exemples, nous pouvons voir deux protocoles différents pouvant partager le même bus. Tout d’abord, nous pouvons voir un contrôleur essayer de lire des octets sur un esclave i2c. Dans la seconde on peut voir un paquet CAN. Il est intéressant de noter que les messages CAN sont envoyés en utilisant uniquement la couche P. Cela montre la flexibilité de SHER-Bus dans la gestion de différentes applications. SHER-Bus peut utiliser des méthodes d'adressage héritées au lieu de son BPI. Cela montre également l'avantage d'avoir un masque sur l'ensemble du paquet au lieu d'une région spécifique. L'adressage peut changer de place ou de longueur dans le paquet, mais tant que l'expéditeur et le destinataire l'acceptent, cela fonctionnera. Les autres passagers du bus ignorent tout simplement le message.