SHER-Bus Pre-specifcation

> **Warning**
>
> Content is not fixed and suject to change without notice!

```
SHER-Bus Stand for:

Systemwide Hub for Efficient Routing Bus

and

SHER-Bus Handles Extensive Resource Bridging, Unifying Systems
```

What is SHER-Bus?

It's a serial bus design to aggregate many low speed serial bus found in modern electronics to one/multiple high sppeed differential(s) pair(s) . Low speed serial bus examples are SPI, I2C, CAN, PWM, PCM, etc. Gones are the day when you realize that you don't have enough i2c bus or UARTS.

This bus can also support more general usage like, humain-machine interface (keyboard,keypad,game controller,joysticks), audio (I2S, SPDIF), low-resolution video (monochome oleds,spi tft lcd), motor control (stepper), General Purpose Input Output (GPIO), Battery managing (SOC, DOD, SOH, Series, Parallel, Charge/Discarge Current, Voltage, Temperatures) etc.

SHER-Bus is also great for general purpose communication (like UART , JASON, protobuf, modbus, etc) between controller.

Why SHER-Bus?

Sherbus is born of the frustration of working with the many, bloated, slow or propritary communication protocol in embedded device. Sherbus is lightweight and easy to understand by anyone from the maker to the veteran designer. The protocol is modular (Only the Protocol layer [P] is mandatory). This modularity give the implementers leway to mold the communication to the product and not the otherwise. Implementers won't need to manage a complex stack for something simple (Like for example a full USB CDC class for a basic UART). The complexity of the software is designed to scale up with the complexity of the bus. In other word, easy task are easly done on simple bus. Otherwise, The more feature added (ex, Bus position identification,multiple lane, etc), the more care need to be done to ensure the integirty of the bus. That meant that the bar of entry is verry low, but the bus still powerfull enough to handle complex task when Implementers need it. This is the inovation SHER-Bus hold beacuse current protocols are either too complex for simple task (a USB stack can take most of the progam space on most microcontroller even Tiny-USB) or too simple for complex task. SHER-Bus is both simple for simple task and ressource full for power-user.

Another point as why we need a protocol like SHER-Bus is that as much RISC-V helped open the CPU market to lisence free IP for CPU, many System on chip require the use of proritary IP for its connectivity. One goal of SHER-Bus is that one day a 100% free and open source SOC would hit the market. With the help of SHER-Bus and many more project like this, the open source community can achive this objective.

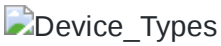
Bus design

> **Warning** MLVDS is not a fixed choice and is suject to test and price comparaison(MLVDS tranciver are expensive!)

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 device (30 controler/bridge and 2 END bridge) can be connected to a single differential lane. 30 device can sound limited but the theorical limit of i2c device on SHER-Bus is 22098![^1] Multiple serial connection (like in a backplane) can be added to increase throughput. The clock is embeded into the datastream so no need to add a clock lane. An optional clock can be added to sychronise function like audio. It use 8bit to 10bit encoding (or manchester idk yet) on the physical layer to ensure DC balance and give a first layer of error checking.

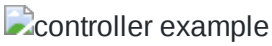
[^1]: 127 i2c device * 6 i2c master port on 1 bridge * 29 bridges (we need at minimum 1 controller) = 22098

Bus Devices

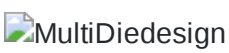


There is only 3 type of device that serve different function in the bus network.

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.



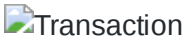
In second there is the bridge which is job is to close the gap between the new bus and the already existing protocol. They can have mutitude of serial bus(up to 6 and 1 general control channel using [BPI]). They consist of mostly ASIC or FPGA. Altough at the time or writing no ASIC or FPGA code as been manufactured/written. Micrcocontroler can also act as bridge. Brige can be integrated into already existing design in die or in multi-die package. Folowing the design once reuse many, it help reduce redesign complexity and speeding up time to market. Bridge implementer need to be careful about licencing of existing bus as some are not free to implement.



In third, There is the End bridge which job is to connect multiple SHER-Bus togehther or other HIGH-SPEED bus (like USB, SDIO or Ethernet). They terminate the bus(where the temination resistor are) that's why they are a limit of 2 that can be connected on the bus. They are complex and not mandatory.

Packet achitecture

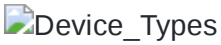
The packet architecture follow the same philosopy taken by the RISC-V cpu achitecture. Only one set of instruction is mandatory(P for SHER-Bus) and each set is labeled by a letter.(ex.: I M C A F D Q for risc-V CPU) or a word/achronim (ex.: Zicsr). What is different because of the nature of communication is that each layer is embeded into the payolad of layers under it. For example a Audio packet (A) is embeded into a Bus Protocol Identification [BPI] which is at his turn embeded into a stream package (S) which is at his turn embeded into a protocol package (P). Whe have structure like P(S(BPI(A))), but if that audio message is adressed to everyone on the bus we can remove the BPI layer and have message structured like this : P(S(A)). This greatly increase the flexibility of the bus. A reciver can perform and AND wise mask to the whole message to see if the message is of interest to him and discard those who aren't (ex.: an I2C brdge dont care about an audio(A) package but an I2S bridge do).



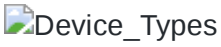
Protocol layer (P)

The protocol layer is the only mandatory layer of the specification. It handle the minimum to be a valid transaction. Implementer use this layer to send very low level message like a point to point UART-like communication [^2] , since adressing is handeled in higher level only point to point or multidrop bus configuration is possible with using only this level. This is a feature because in many aplication you wouldn't want a heavy stack for something simple. This also grant implementer the freeddom to create custom protocol stack for application that arent covered by the existing stack.(ex.: SAE J1939 and CanOPEN are both stack that are built upon the unrestricitve nature of the CAN protocol, SHER-Bus trying to do the same but free with having a common stack help bridge implemter and bus implementer have a common ground to work with). the first byte is to tell if its a stadard packet or a custom one. A one(1) on the MSB of the first byte tell that the payload is a SER-Bus compliant message. Implementer can send custom message by setting the first byte to 0(0x00).

[^2]: only 2 Controller should be connected on the bus. A multipoint UART sould be availlable in the application layer.



Network layer



Contol Messages (C)

Control message are use to change how the bus or a device react. For example an implementer can perform a device reset. It is also used by the [BPI] layer for Dynamic Adressing.

Interrupt Messages (I)

Interrupt Messages are designed to be as close as possible to computer interrupt. They can use the [BPI] 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.

Bomerang Messages (B)

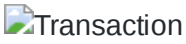
Bomerang Messages, like in their name sugest, are meant to come back to the sender. Uppon reception of a (B) message the reciver use it to perform an action and then send back the same message with modifcation according to that action. The (B) message are an exception because they must support the Bus Position Identification messages, otherwise everyone on the bus would send back the message. Sending back the same message also ensure that it was recived correctly and allow de syncronisation of transaction(the back and forth can happen with other message seprating them). An example would be an I2C read on the bus. A controller coul ask for read of W adress X i2c salve on the Y I2C bus of Z bridge. The bridge would respond after performing the read that he recived n bytes of data from W adress X i2c salve on the Y I2C bus by sending back the same message but swapping the reciver/tranciver byte of the [BPI] and appedning the readed data.

Stream Messages (S)

Stream message are for when data integrity is not important but a constant flow of data is. Stream message are for an audio stream for instance. Its not bad if 1 or 2 packet are lost its better to have a constant flow of packet. Stream Messages are better if they are on a separate bus as they are the lowest priority.

Bus Position Identification (BPI)

The Bus Position Identification give controler a way to talk with a specific device while other remain unchanged. Bridge must support BPI because they never know what bus the will be part of. Each device can have up to 7 sub-device. sub-device 0 is reserved for control or when sub-device are not used. Device can get an adress of 3 way possible: Statically, Pseudo-Dynamicly and Dynamicly.



Static Adressing

Each device on the bus has a pre determined adress that dose not change. It is the resposibility of the Bus implementer to set an unique the adress to each device present on the bus. This mode of adressing is useful when the implementer know the bus layout and it dosen't change like in a fully closed system with no exterior ports. To be documented.

Pseudo-Dynamic Adressing

Each device get his adress with using an external way. For eample in a backplane each slot is labeled electronicaly(gpio or i2c eeprom) with an unique adress. SHER-Bus device read that adress upon power up and use it for communication. To be documented.

Dynamic Adressing

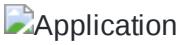
Each device get his adress asking the SHER-Bus using Control(C) message. To be documented.

High level Messages

> **Warning** To be written

The high level message are function that the bus can take. From the transport legacy protocol to a function based ccommunication (Battery management, Data logging, Video, Audio) SHER-Bus can do it. Each

application is defined by a Aplication code that is given by the BUS community.(TO BE Deterined)



Transaction Example

In those example we can see two different protocol that can share the same bus. Firstm, we can see a controller try to read bytes off a i2c slave. In the second we can see a CAN packet. Its interesting than CAN message are sent using only the P layer. This show the SHER-Bus flexibility at handleling different aplication. SHER-Bus can use legacy adressing method instead of it's BPI. This also show the power of having a mask on the whole packet instead of a specific region. The adressing can change place or length in the packet but as long as the sender and reciver agree to it. The other on the bus simply ignore the message.