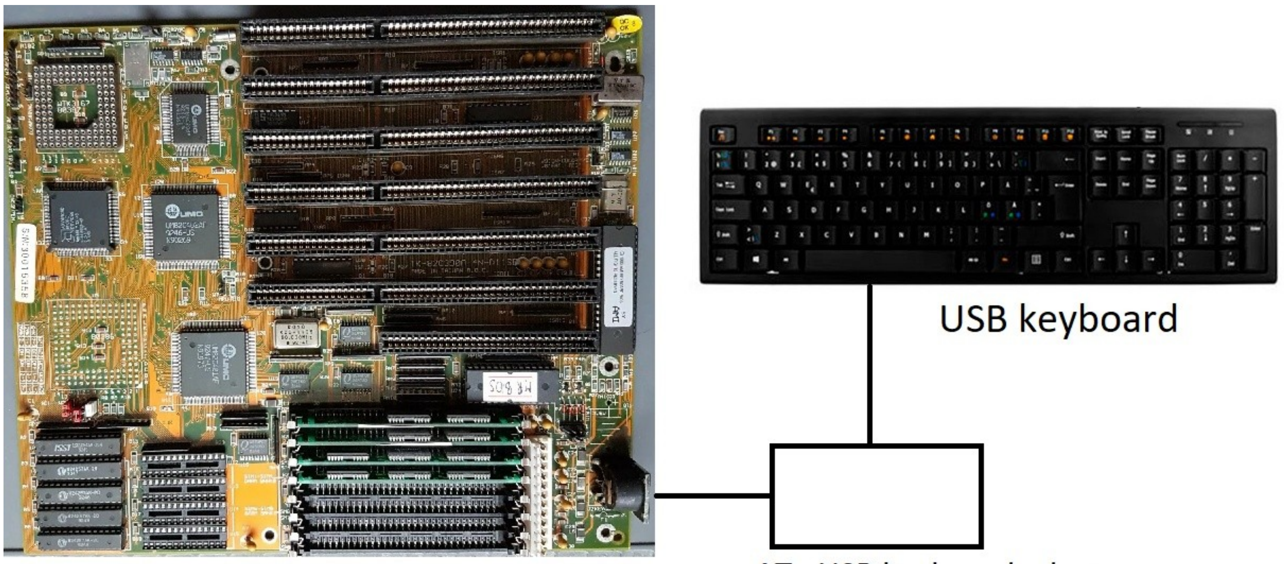
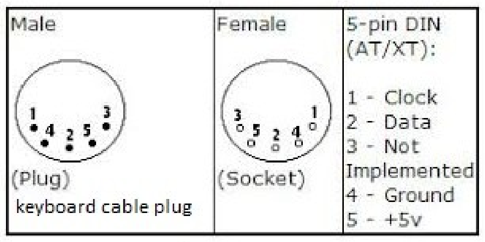
This project is to make an adapter to use any USB keyboard with an IBM AT PC compatible motherboard. The adapter emulates an AT keyboard seen from the motherboard, and emulates an USB HID keyboard host seen from the USB keyboard.



Motherboard with AT keyboard connector AT - USB keyboard adapter

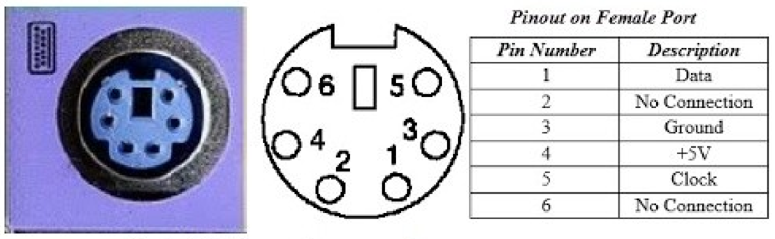
The IBM PC AT keyboard interface and protocol was specified in 1984. The AT motherboard keyboard connector is a 5 pin DIN socket and it uses the data communication protocol known as the PS/2 keyboard protocol. The IBM PC XT uses a similar keyboard connector, but with another communication protocol. A dedicated PS/2 keyboard connector came out in 1987. The keyboard connector pinout for the IBM AT/XT and PS/2:



5-pin DIN (AT/XT):

1 - Clock
2 - Data
3 - Not Implemented
4 - Ground
5 - +5v

IBM AT/XT connector



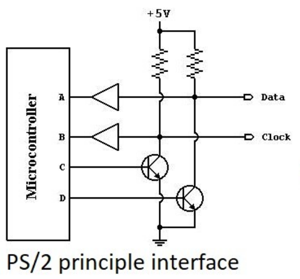
Pinout on Female Port

Pin Number	Description
1	Data
2	No Connection
3	Ground
4	+5V
5	Clock
6	No Connection

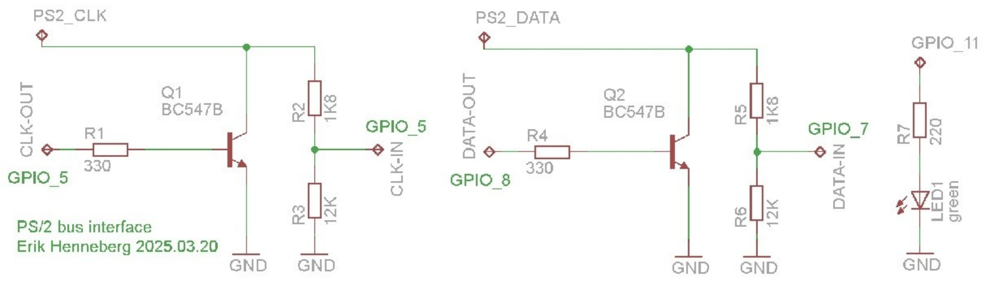
PS/2 keyboard connector

Some older USB keyboards are able to work with both the PS/2 and the USB HID keyboard standards, and using an AT-PS/2 adapter plus a PS/2-USB adapter can do the AT-USB keyboard conversion, but I want to be able to use any USB keyboard.

The AT (or PS/2) electrical interface uses half duplex communication between the motherboard and the keyboard:

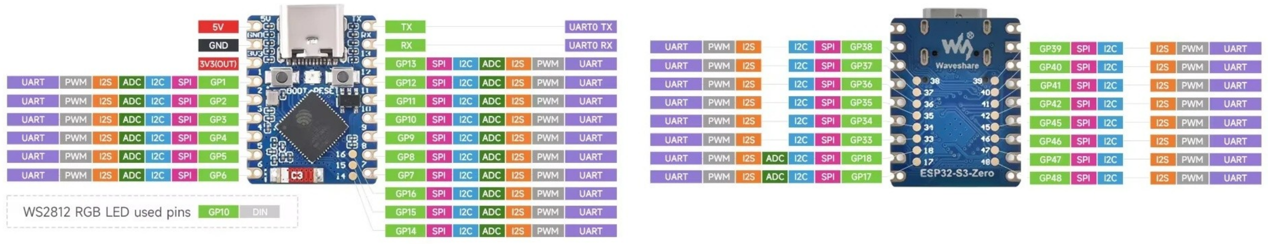


PS/2 principle interface



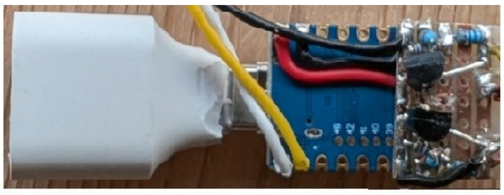
The PS/2 electrical interface principle on the left is used at the keyboard as well on the motherboard. Two open collectors for the outputs and two buffers as input with the pull up resistors in the range of 4K7..10K ohm. On the right is the schematic for the interface for the ESP32-S3 microcontroller I use. The GPIO-x pin names refer to the microcontroller I/O pins:

For the USB interface to the USB keyboard , I use the ESP32-S3 zero board that can work as an USB HID host:



A piece of Vero-board was used to hold the interface components, except for R7 and LED1:

Top view with PS/2 I/F


Bottom: PS/2 I/F + USB-C to USB-A OTG

ESP-01 programmer, wiring to UART0

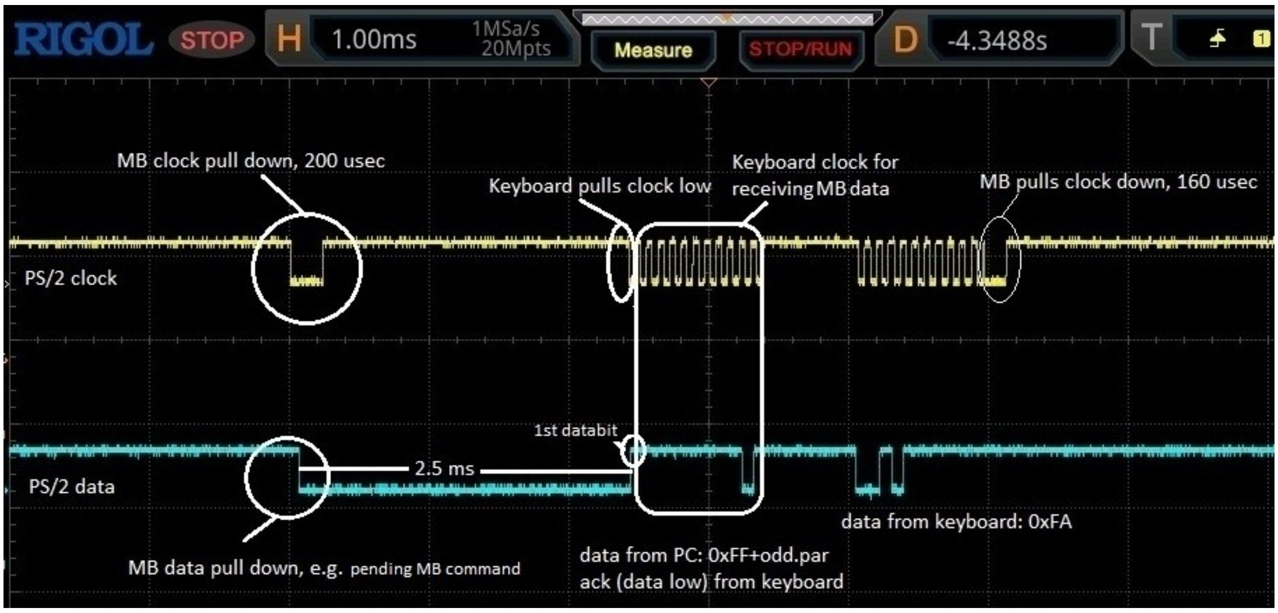
The ESP32-S3 zero GND and 5V are connected to GND and 5V at the motherboard AT keyboard connector. Resistor R7 and the green LED1 are used to indicate a key pressed at the USB keyboard. I use the Arduino IDE and the ESP-01 programmer to flash the code via the microcontroller UART0. Furthermore, an USB-C to USB-A OTG adapter is needed.

When using the electrical interface for the PS/2 half duplex bus some precautions must be taken:

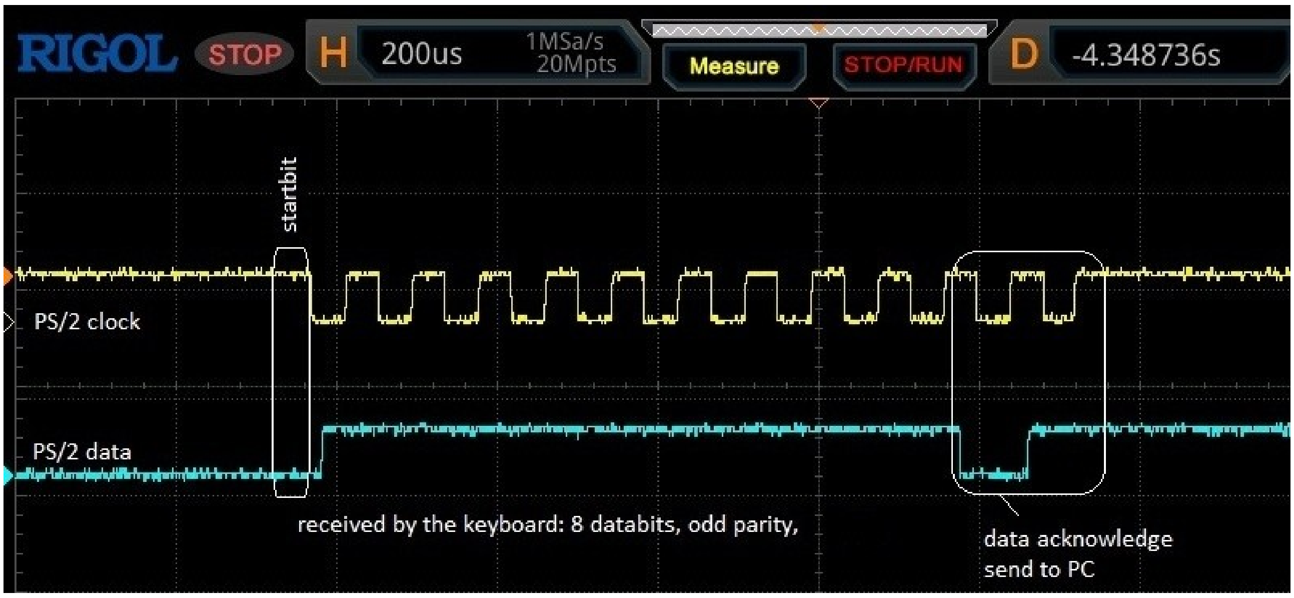
1. The clock signal is always issued by the PS/2 device when data transfer takes place between the PS/2 device and the motherboard. The clock signal frequency in this project is 12.5 kHz , e.g. a clock period of 80 usec.
2. The motherboard can pull the clock signal to low state at any time to notify a busy state or a pending command to the PS/2 device.
3. In case of a pending motherboard command that need to be read by the PS/2 device: The motherboard pulls the clock signal to low state for more than 100 usec. and the motherboard pull the data signal to low state before releasing the clock signal (to high state). At this point , the PS/2 device can clock in the command data from motherboard.
4. Commands from the motherboard the PS/2 device goes on from the BIOS during start-up, from DOS and from system test utilities.

The PS/2 device mentioned above is a PS/2 keyboard or in this case, the AT-USB keyboard adapter, that emulates a PS/2 keyboard way of working.

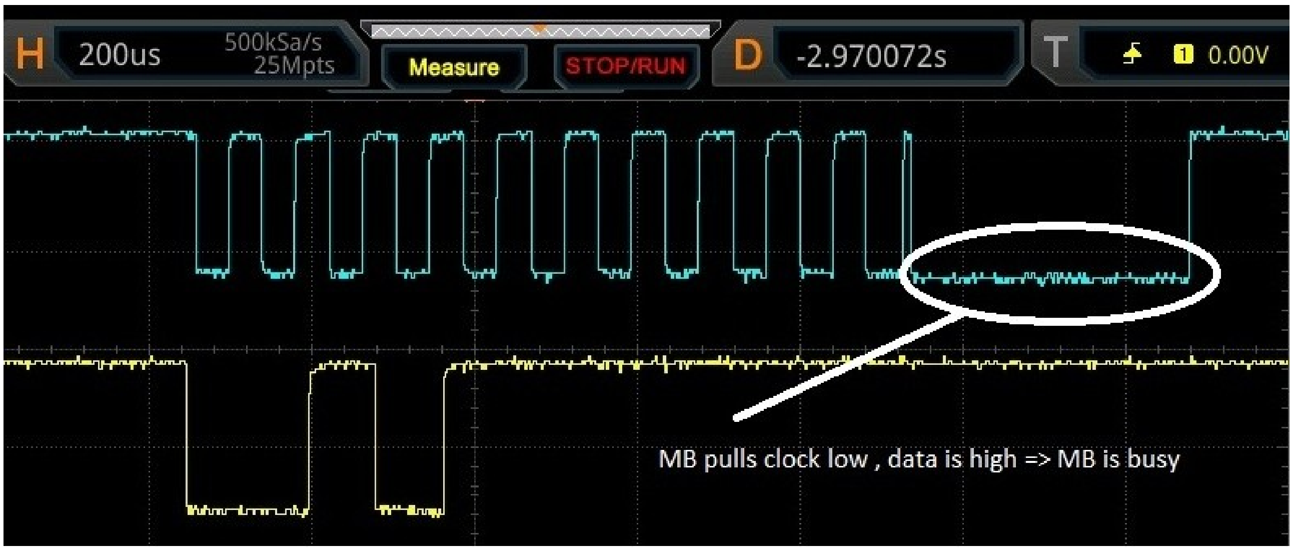
In the timing example below, the motherboard pulls the clock signal low for 200 usec and the data signal is pulled low as well ahead of the rising edge of the clock signal => a pending command is to be read by the PS/2 device. After receiving a motherboard command, an acknowlegde byte (0xFA) is send back to the motherboard:


RIGOL
STOP
H 1.00ms
1MSa/s 20Mpts
Measure
STOP/RUN
D -4.3488s
T
MB clock pull down, 200 usec
Keyboard pulls clock low
Keyboard clock for receiving MB data
MB pulls clock down, 160 usec
PS/2 clock
1st databit
2.5 ms
PS/2 data
data from keyboard: 0xFA
MB data pull down, e.g. pending MB command
data from PC: 0xFF+odd.par
ack (data low) from keyboard

Timing details on receiving the command (resetKeyboard = 0xFF) from the example above. The command startbit is ready on falling edge of the clock. The command has 8 databits and an odd-parity bit. All bits are read by the PS/2 device on the center point of the clock high signal. The PS/2 device returns an acknowledge pulse by pulling the data signal low for one clock period:



Timing details for the PS/2 device to send data to the motherboard, in this case the Acknowledge (0xF2) in response to a received command. All bits are read by the motherboard on the falling edge of the clock signal, e.g a startbit, 8 data bits, an odd-parity bit and 1 stopbit. Note "the motherboard is busy" notification when the motherboard pulls the clock signal low for more than 100 usec., while leaving data signal high when releasing the clock signal.



The PS/2 keyboard commands and scan codes are described here: PS/2 command list . The command (0xFF = reset keyboard) is commonly used, but the use of commands depends on the actual motherboard.

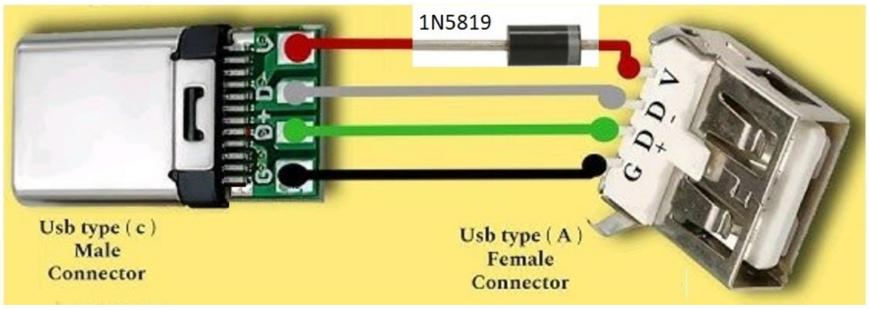
The PS/2 timing for this project is software based, with simple delays for sending clock and data, and an edge triggered interrupt routine.

The interrupt routine is called on a falling clock edge and on a rising clock edge. The interrupt uses an ESP32 timer to measure the time between a falling clock edge and the following rising clock edge. The interrupt routine is also called during transmission of data to the motherboard.

The clock signal is low for 40 usec during the data transmission, but if the clock signal is low for more than 100 usec., it must be the motherboard that pulls the clock low , either for reason of busyness or a pending command. The interrupt routine sets a global boolean in case of a pending motherboard command.

As the clock edge triggered interrupt consumes time both on the falling and rising edges , the delay for the clock low timing must be adjusted to take the interrupt time consumption into account.

Hardware for the USB keyboard interface is an USB-C USB-A OTG adapter , or a homemade version. The reason for the 1N5819 Schottky diode is to block 5V to the motherboard in case an USB sharing switch is used with the USB keyboard.



Software for the USB HID keyboard host mostly rely on this ESP32 implementation: gitHub ESP32 USBhost . I made some changes to get the adequate information in order to convert an USB keycode to a set of PS/2 keyboard scan codes to be send to the motherboard.

Here is the gitHub repository for the Arduino
software: https://github.com/ehenneberg/ESP32_USBhost_AT_keyboard

Made a adapter test video to be seen here: AT-USB_Keyboard_test