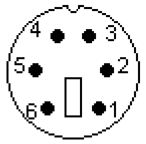
Author: Giorgos Lazaridis

The PS2 protocol

Overview of the PS2 connection

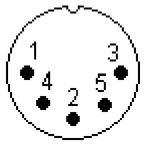
The keyboard that most of you have in front, is using the IBM standard protocol to communicate with your computer. This protocol has the responsibility to send the key scan codes that you press to the pc and get some response commands from it. This means that we are dealing with a bi-directional type of protocol as each device (pc/keyboard) sends and receives commands.

At first, we will take a look at the pin-out of the ps/2 connector (actually it might be also a 5 pins connector but is the same as a ps/2).



The PS2 connector

1. KBD Clock
2. GND
3. KBD Data
4. N/C
5. +5V (VCC)
6. N/C



The PS2 5pins connector

1. KBD Clock
2. KBD Data
3. N/C
4. GND
5. +5V (VCC)

The protocol

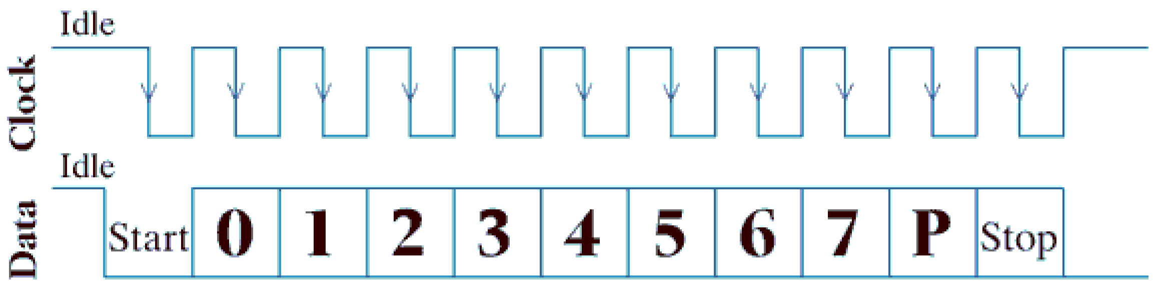
As said before, the protocol will take over the communication. This is done using the data and clock lines. These lines are **High** when no communication takes place **(idle)**. The communication from keyboard to host and from host to keyboard are slightly different so will be explained separately. Keep in mind that the frequency of the clock is about **20-30Khz** and is generated **by the keyboard** but **only when a transmission takes place**.

From the keyboard to the host

The keyboard is free to send data to the host when **both Data and Clock lines are kept high**. The keyboard will take the Data line low (Start bit) and then start generating the clock pulses on the Clock line. Each bit is sent in series with the following order:

Start bit => 0...7 data bits => Odd parity bit => Stop bit. (8/O/1)

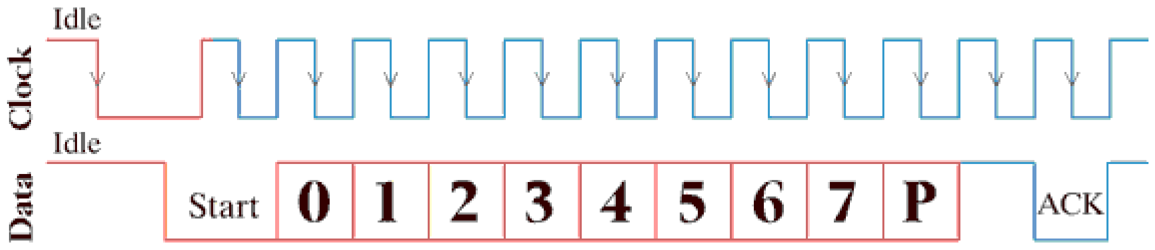
Each bit is read on the falling edge of the clock so keep it synchronized correctly as seen on the following diagram:



The host has priority number 1 against the keyboard. This means that it can send data to the keyboard whenever it wants. So, before you start sending data to the host **make sure it is not ready to transmit to the keyboard** (the Clock line must be High). Otherwise both data will be lost.

From the host to the keyboard

The host will take the Clock line Low for about 60uSec. This is more than one bit length and thus the keyboard is prevented from sending data to the host. Then, the Data line will be taken into Low state and the Clock line will be released by the host. The keyboard must now start generating the clock pulses. All data are read from the keyboard after every falling edge excluding the first one that is for synchronization purposes. After receiving the parity bit, the keyboard will send another one pulse and then take the Data line into Low state for the next pulse. This is for acknowledging the reception of new data. If the Data line is not idle (High state) after the reception of the tenth bit, the keyboard will continue sending clock pulses as new data will arrive by the host.



The red line show the state changes driven from the host and the blue line are changes driven from the keyboard.

Sending a key

Now what happens when you press a key? Simply, the keyboard will send the scan code of the key that you pressed to the host. The keyboard distinguish two states for the key: the **key down state**and the **key up state**. What does this means? The keyboard will send the scan code for the key that you pressed (key down) and when you leave the key up it will send an '**F0**' following by the scan code of the key (key up). An exception is made when the key-down state was sent for an extended scan code key. If a key with two bytes scan code (E0.....) was pressed, upon release the keyboard will first send the E0, then the F0 and then the other half of the key scan code. So use extra caution when programming.

For example, if you press the 'A' key, the scan code**1C** will be send. When you leave the key, the **F0 1C** will be sent. If you press the 'A' key and you keep it pressed more than it's typematic delay, the keyboard will keep sending the key's scan code according to it's typematic rate, until the key is released.

The keys do not have simply one byte scan codes. It could be possible as one byte can hold up to 255 different keys but it is far from true. The extended keys have two bytes and the first isE0. As for the pause/break key... the scan code is **E1 14 77 E1 F0 14 F0 77**.