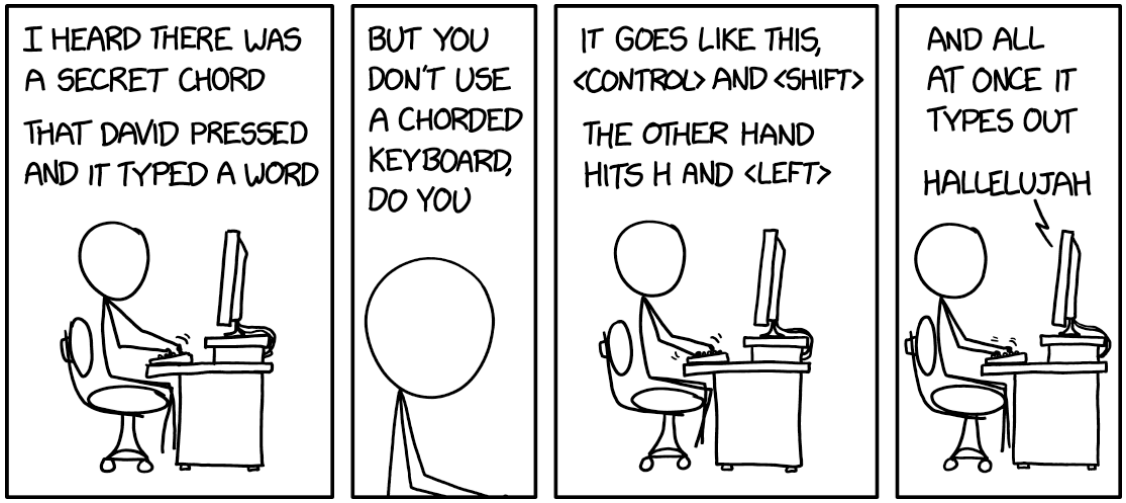
Admin

Lab 5

Bring in code for show & tell



https://xkcd.com/2583/

Today: PS/2 protocol

Reunite with our first and oldest friend, gpio module!

1987 called and asked for its keyboard back

Road map

- gpio
- timer
- uart
- strings
- printf
- backtrace
- symtab
- malloc
- keyboard
- shell
- fb
- gl
- console

UART

Bi-directional communicate laptop <-> Pi (**`uart_putchar/getchar`**)

`8N1`: start bit, 8 data bits, no parity bit, 1 stop bit

Asynchronous: no clock, need precise timing both ends

- *What can happen if drift/inaccuracy between sender/receiver clocks?*

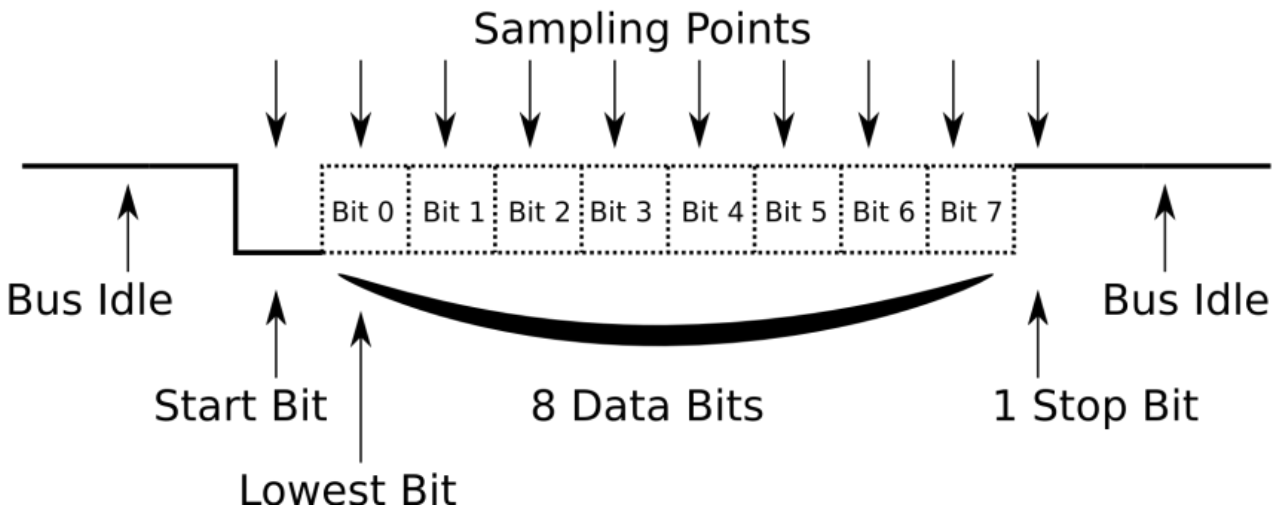


Figure from http://old.myhdl.org/lib/exe/detail.php/projects:rs232image.png?id=projects%3Auart_rs232_receiver_transmitter

PS/2 interface

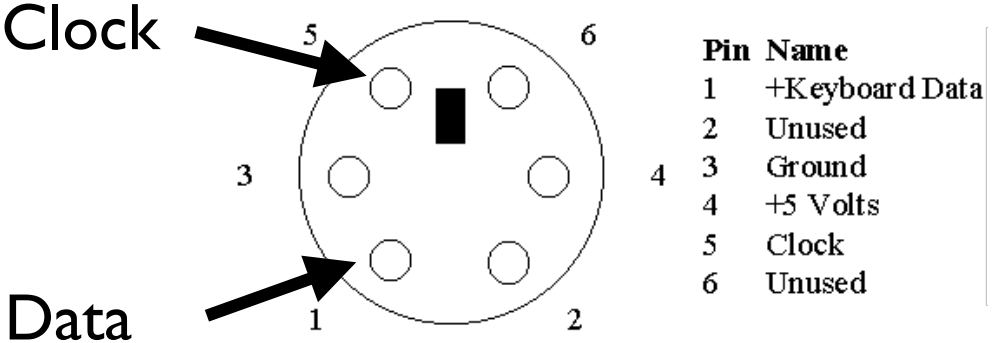
PS/2 is the original serial protocol for keyboards and mouse (since replaced by USB)

Computer PS/2 ports

http://www.computerhope.com

6-pin mini-DIN connector

PS/2 Keyboard and Mouse Cable



Pin	Name
1	+Keyboard Data
2	Unused
3	Ground
4	+5 Volts
5	Clock
6	Unused

Cable (male) pinout

PS/2 Protocol

- **8-Odd-1**
  - Start, 8 data bits, odd parity, stop (11 total bits)
- Synchronous, clocked by sender
  - Two lines: clock and data
  - Pulse clock to indicate when to read data line
  - Read happens after falling edge of clock
- Open-collector (both clock and data)
  - Circuit is high when idle
  - Pull low (connect to ground) to start signal

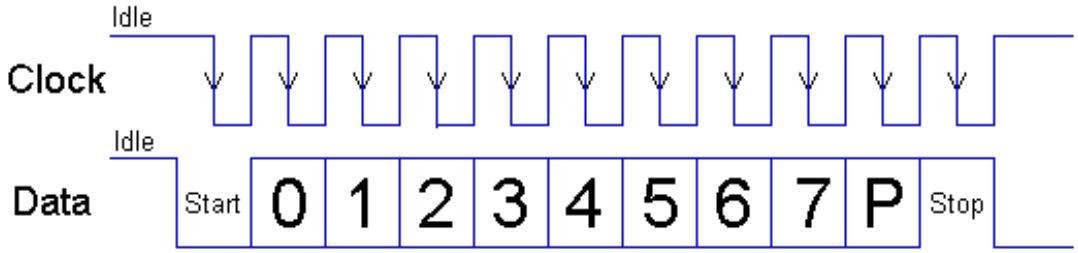
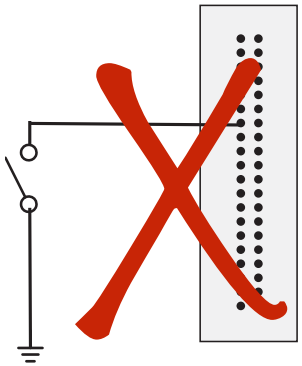
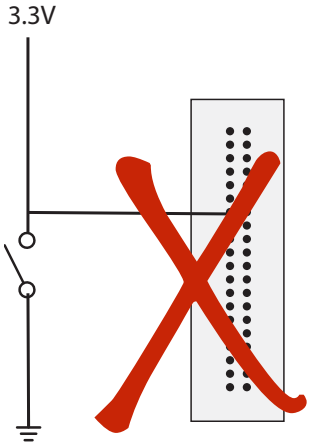


Figure from http://retired.beyondlogic.org/keyboard/keyboar11.gif

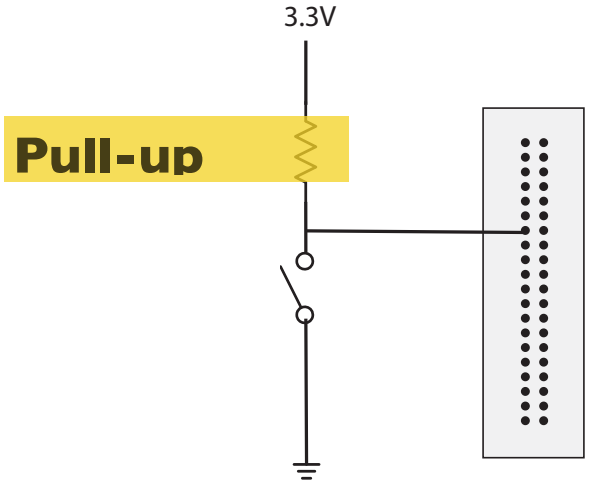
Switch

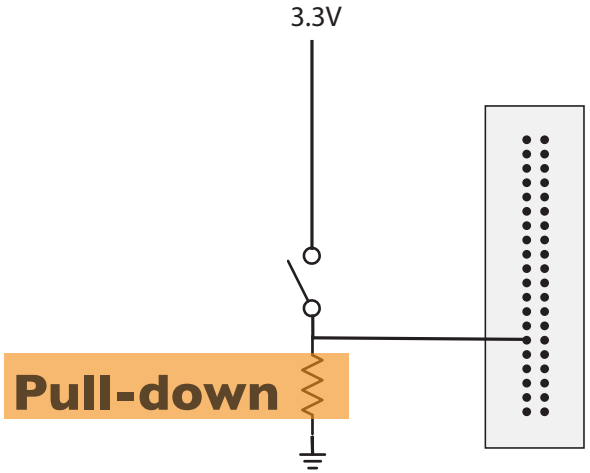


Wired as above,
what does switch read when open?



Wired as above,
what happens when switch is closed?

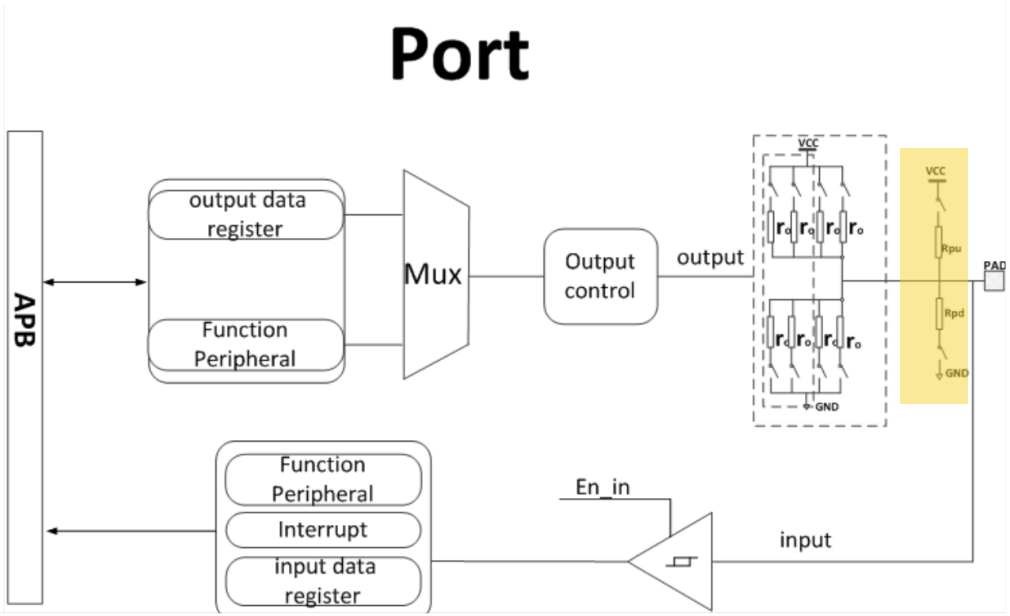




How does adding pull-up or pull-down fix the problems from above?

Software-controlled pull state

Figure 9-67 GPIO Block Diagram



9.7.4 Register List

Module Name	Base Address
GPIO	0x02000000

Register Name	Offset	Description
PB_CFG0	0x0030	PB Configure Register 0
PB_CFG1	0x0034	PB Configure Register 1
PB_DAT	0x0040	PB Data Register
PB_DRV0	0x0044	PB Multi_Driving Register 0
PB_DRV1	0x0048	PB Multi_Driving Register 1
PB_PULL0	0x0054	PB Pull Register 0
PC_CFG0	0x0060	PC Configure Register 0

See our
`gpio_extra.h`

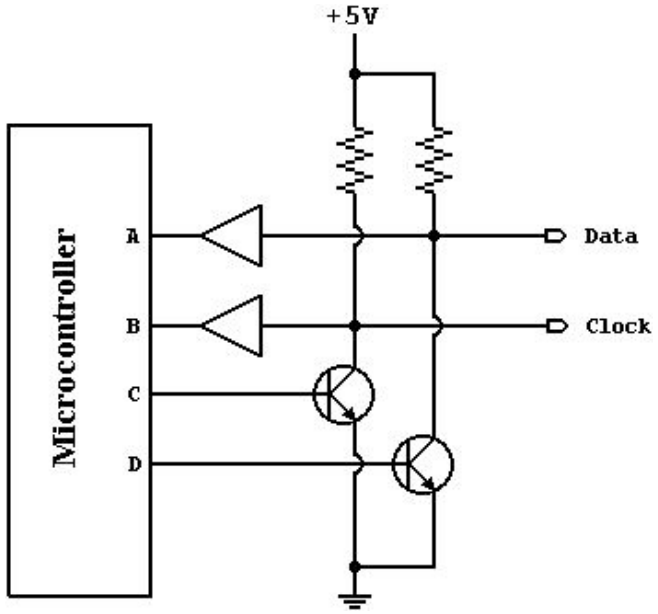
High-impedance, the output is float state, all buffer is off, the level is decided by external high/low level. When high-impedance, the software configures the switch on Rpu and Rpd as off, and the multiplexing function of IO is set as IO disable or input by software.

Pull-up, an uncertain signal is pulled high by resistance, the resistance has a current-limiting function. When pulling up, the switch on Rpu is conducted by software configuration, the IO is pulled up to VCC by Rpu.

Pull-down, an uncertain signal is pulled low by a resistance. When pulling down, the switch on Rpd is conducted by software configuration, the IO is pulled down to GND by Rpd.

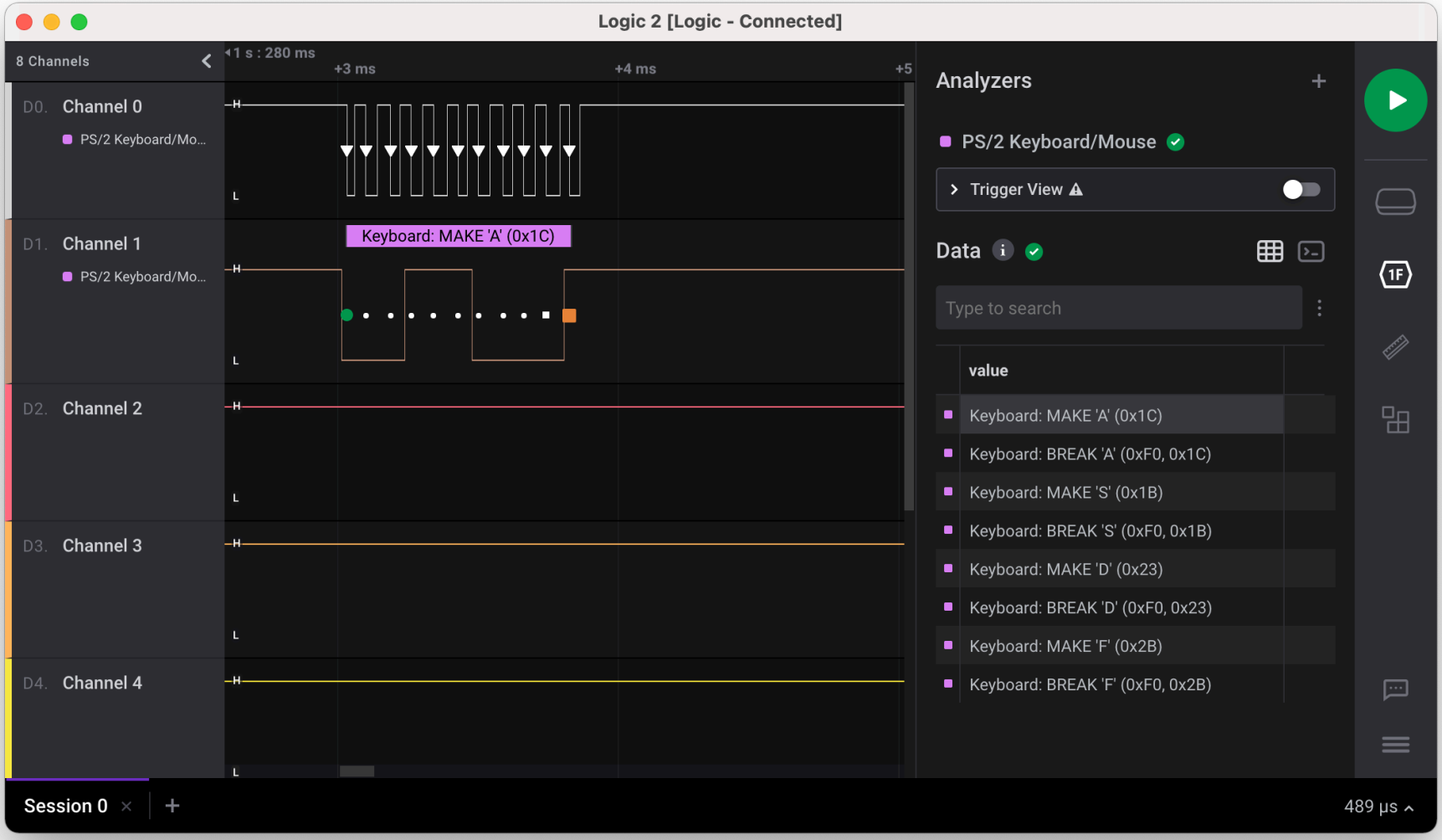
Ref: D1-H User Manual p.1093

Open collector



- Clock and Data lines are *pulled up* to 5V in idle state
- Switching on transistor sets line to 0V
- Communication bi-directional (keyboard or Pi can pull line low to take control)

PS/2 logic analyzer demo

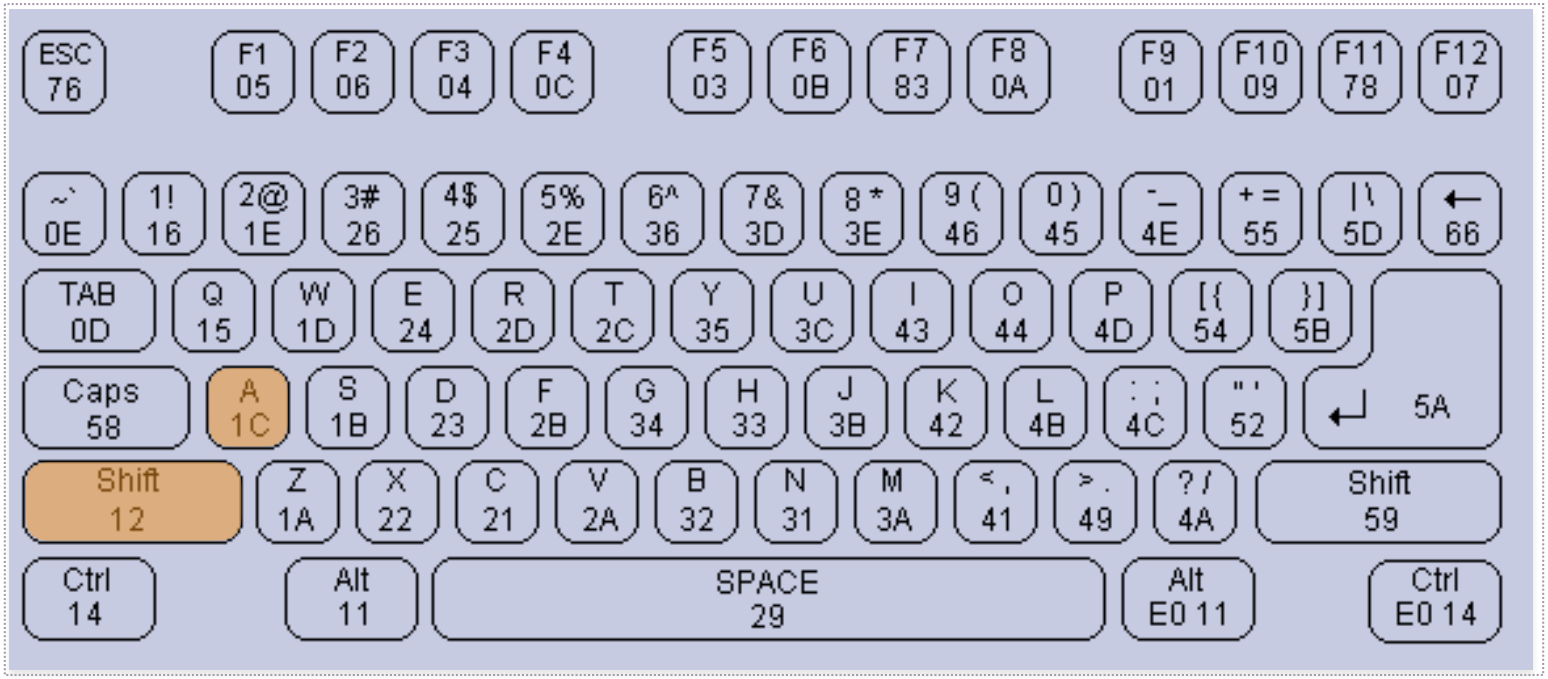


Code to read scancode bits

code/ps2/

PS/2 Scancodes

http://www.computer-engineering.org/ps2keyboard/



`0xF0` is "Break" scancode

Key	Action	Scancode
A	Make (press)	**`0x1C`**
A	Break (release)	**`0xF0 0x1C`**
Shift (left)	Make (press)	**`0x12`**
Shift (left)	Break (release)	**`0xF0 0x12`**

Keyboard read scancode

code/scancode/

Parity bit

Even parity: XOR 9 bits (8 data, 1 parity), result even (count of on bits is even)

Odd parity: XOR 9 bits (8 data, 1 parity), result odd (count of on bits is odd) **PS2 protocol is odd parity**

even

data	data	data	data	data	data	data	data	parity
1	1	0	1	0	1	1	0	1

odd

data	data	data	data	data	data	data	data	parity
1	1	0	1	0	1	1	0	0

Error recovery

When reading PS/2 scancode, check for errors:

- Start bit != 0
- Parity not odd
- Stop bit != 1
- Time between bits is too long

When detect error, discard partial bits and restart read

tekkieneet says:
March 11, 2021 at 1:01 pm

While simple logic to decode the PS/2 protocol, it is unlikely it can recover gracefully glitches/ESD/accidental connector removal/reconnection. When clock bits are missed without a resynchronization, all the data collected from that point are garbled. This was one of the things why the early PC don't handle reconnect well and requires a reboot if someone tripped on the keyboard cable.

To recover, you would need a timeout on last clock pulse and try to resynchrize the start bit. I have implement that on my PS/2 code and it always recovers.

https://hackaday.com/2021/03/11/decoding-the-ps-2-keyboard-protocol-using-good-old-fashioned-hardware/

Key (scancode) ≠ character

- Scancode identifies key, not ASCII value
  - e.g A key sends scancode `0x1c`, ascii 'A' is `0x41`
  - Typically keyboard has 104 keys, 127 ASCII character codes
- Extra keys
  - Special keys - handled by code in keyboard driver/application
    - Function keys, arrows, delete, escape, ...
    - Modifiers (shift, control, alt, command)
  - Multiple keys with same function
    - Left and right shift
    - Numbers on keypad vs. keyboard

Keyboard Viewer

Modifier keys

None

[Shift]

[CapsLock]

[CapsLock][Shift]

Layered abstractions

https://cs107e.github.io/header#keyboard

`uint8_t keyboard_read_scancode(void)`

Read single well-formed scancode (8-bit unsigned int)

`key_action_t keyboard_read_sequence(void)`

Read sequence of scancodes corresponding to single key press or release

`key_event_t keyboard_read_event(void)`

Return key event including modifier state

`char keyboard_read_next(void)`

Return typed ASCII character