

Keyboards

The PS/2 Protocol

Debugging

Always start from a known working state; stop in a working state. If it breaks, what changed?

Take a simple small step, check it carefully, then take another small step. Programming is much simpler if you figure out a good order to write the code!

Make things visible (printf, logic analyzer, gdb)

Fast prototyping, embrace automation, 1-click build

Systematic (D&C), not random, search for bug. Form hypotheses and perform experiments

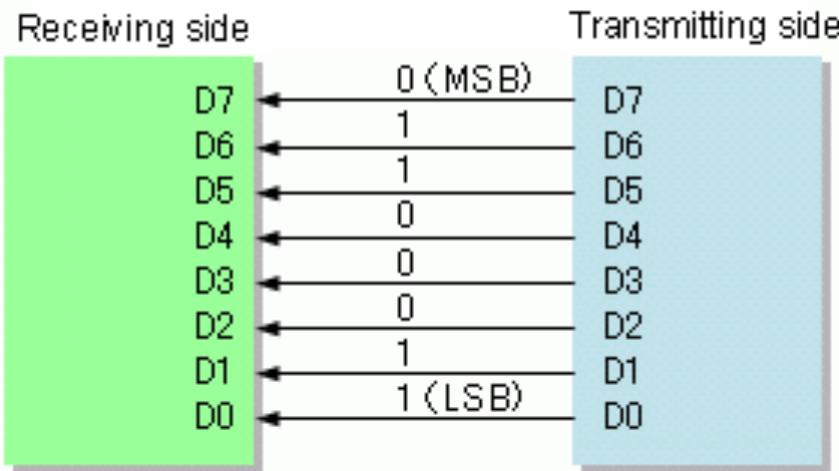
Don't be frustrated by bugs, relish the challenge

gpio
timer
uart
printf
malloc
keyboard
fb
gl
console
shell

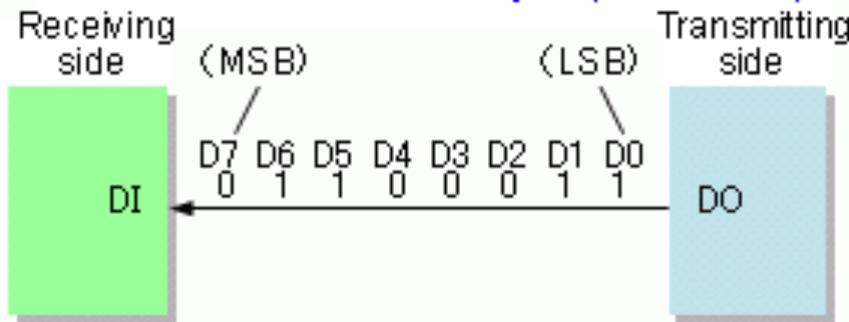


Serial vs. Parallel

Parallel interface example

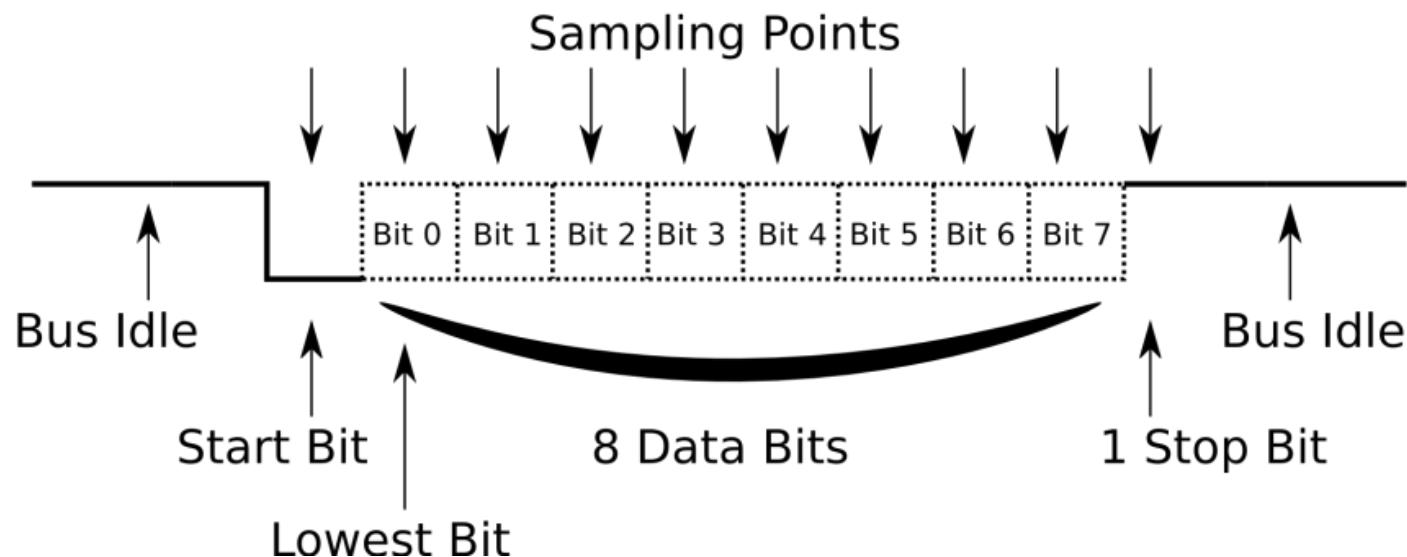


Serial interface example (MSB first)

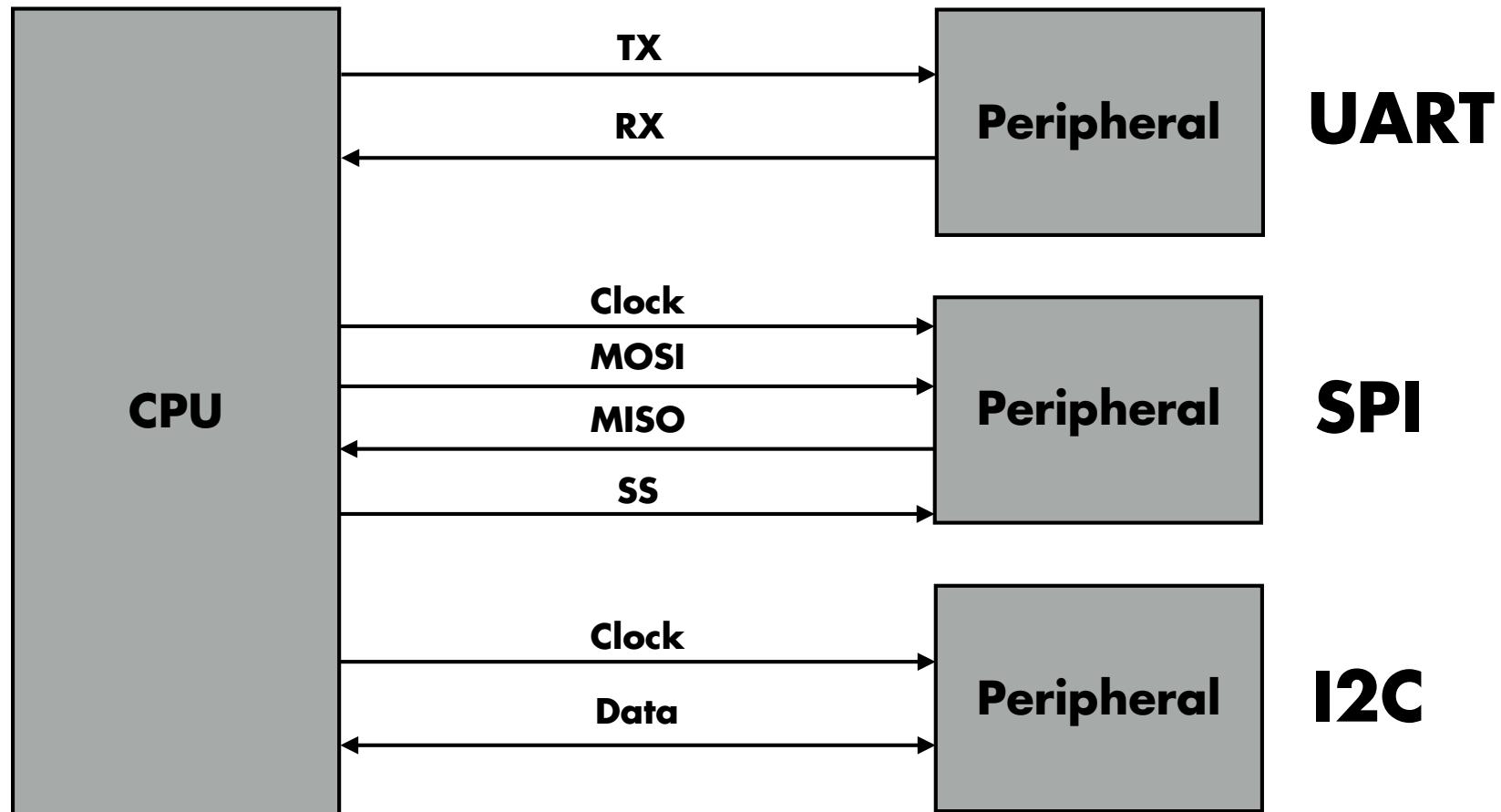


UART

- **Used in your printf & the bootloader**
- **Start bit, data bits, (0 or 1), stop bits**
- **No clock, requires precise timing**



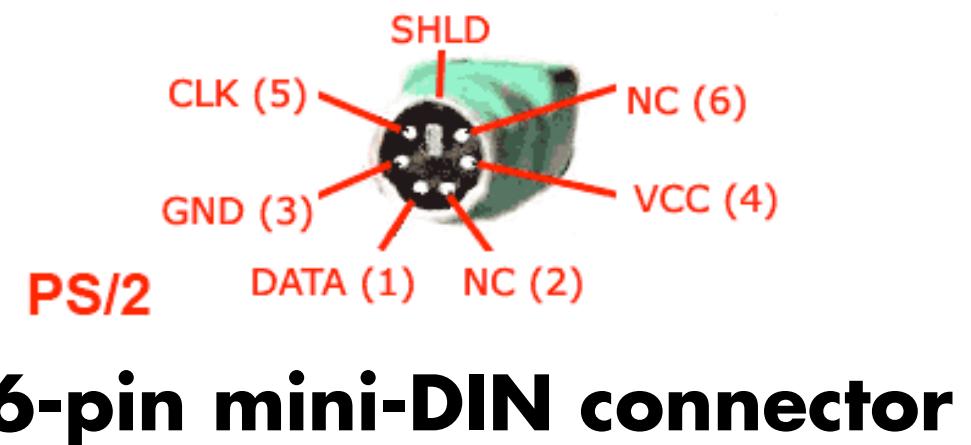
Synchronous vs Asynchronous Serial Protocols



Synchronous has a clock

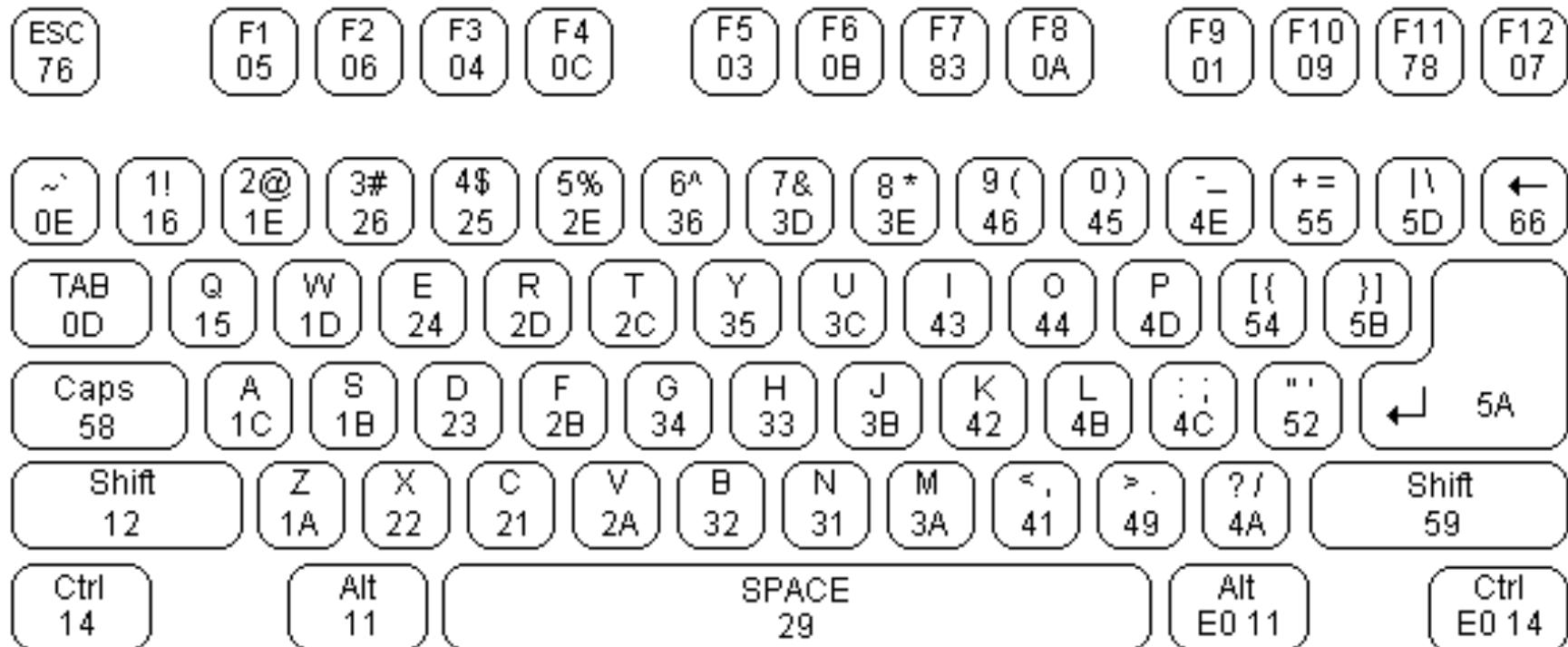
PS/2 Keyboard

- PS/2 is an old serial protocol for keyboards (replaced by usb keyboards)
- Synchronous: CLK and DATA lines



Keyboard Scan Codes

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



Make (press) and Break (release) codes 0xF0

Key	Action	Scan Code
A	Make (down)	0x1C
A	Break (up)	0xF0 0x1C
Shift L	Make (down)	0x12
Shift L	Break (up)	0xF0 0x12

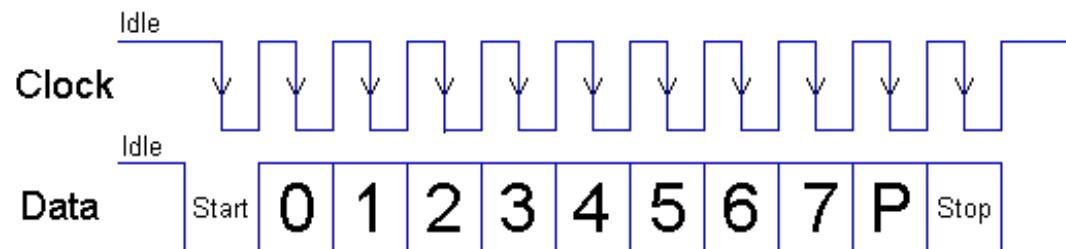
Keyboard Scan Code Demo

PS/2 Protocol

Data changes when clock line is high

Read data when clock is low

Payload: start bit, 8 data bits (lsb-first), parity bit, 1 stop bit (11 total)



<http://retired.beyondlogic.org/keyboard/keyboar1.gif>

PS/2 Protocol Demo

(logic analyzer)

Keys (Scan Codes) ≠ Characters

Characters are printable

ASCII control codes: tabs, form feeds, SOH, NAK

Special keys - interpreted by the OS or App

- F1, ..., F12
- Arrows, insert, delete, home, ...

Keys with duplicated functionality

- Left and right shift, ...
- Numbers on keypad vs. keyboard



Keys ≠ Characters

Modifier keys

- [SHIFT] [OPTION/ALT] [CTRL] [CMD] [FN]



Keyboard Viewer

Keys ≠ Characters

Modifier keys



[Shift]

Keys ≠ Characters

Modifier keys



[Caps Lock]

Keys ≠ Characters

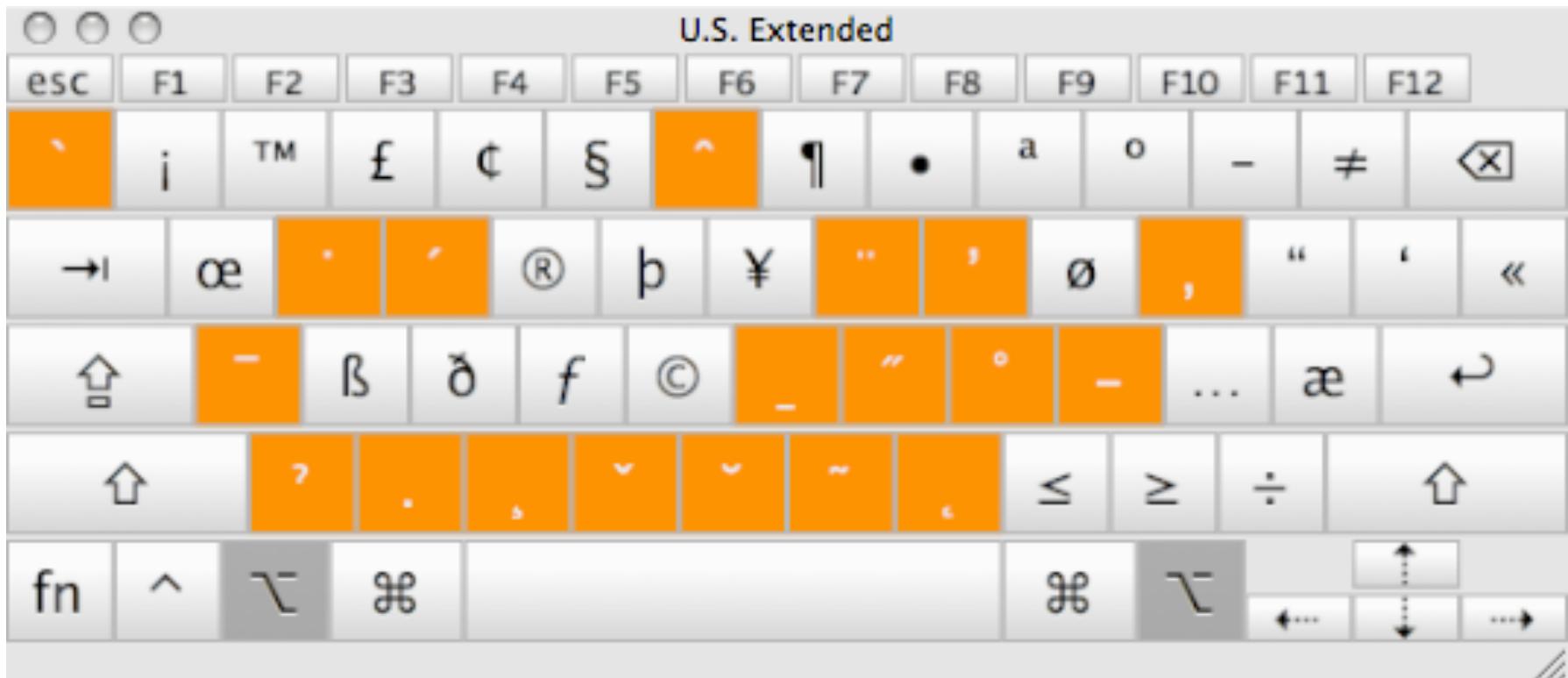
Modifier keys



[Shift + Caps Lock]

Keys ≠ Characters

Modifier keys



[Option]

Orange = Dead Keys

Keys ≠ Characters

Modifier keys



[Option-A]

Layers (keyboard.h)

```
unsigned char keyboard_read_scancode(void)
```

- returns scancode

```
int keyboard_read_sequence(unsigned char seq[])
```

- returns sequence of up to 3 scan codes (with PS2_CODE_RELEASE=0xF0,
PS2_CODE_EXTEND = 0xE0)

```
key_event_t keyboard_read_event(void)
```

- keeps track of modifier keys
- returns key_event struct : scancode, modifiers, action, ...

```
unsigned char keyboard_read_next(void)
```

- maps key events to ASCII characters
- returns ASCII character, modifier and special keys >= 0x90

Parity Bits

Parity = XOR of data bits

Even/odd parity: an even/odd number of 1s (including parity bit)

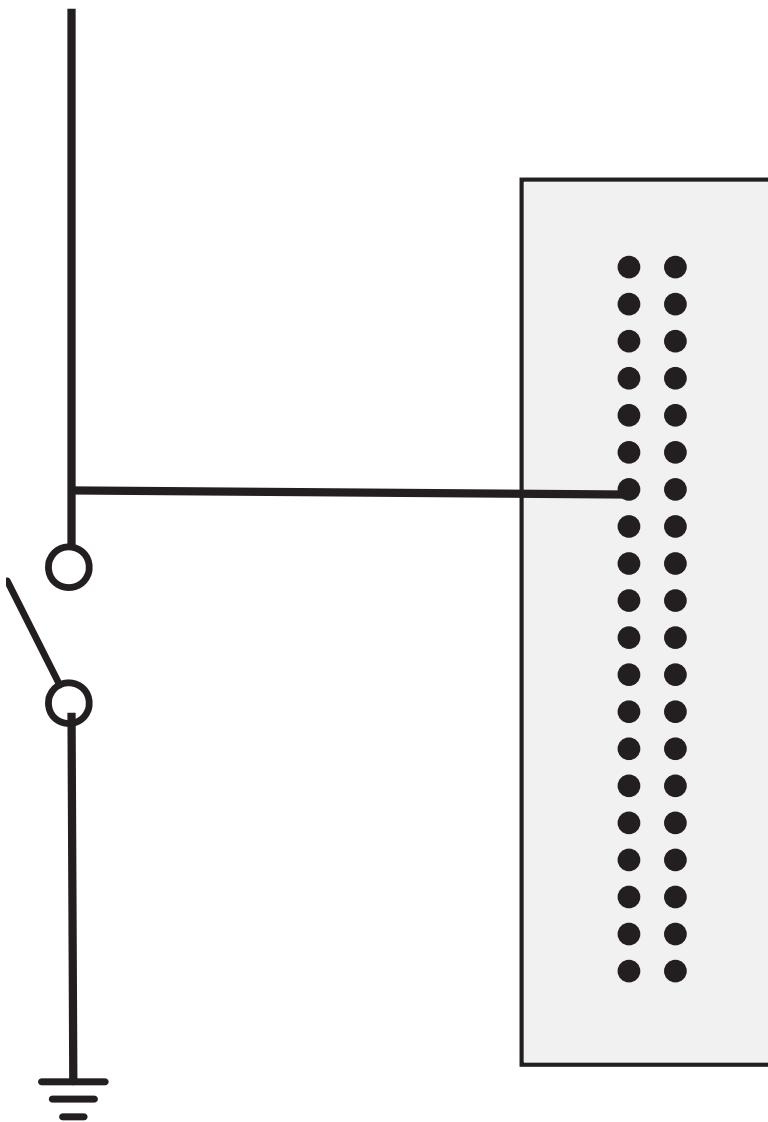
even	data	parity						
	1	1	0	1	0	1	1	0
odd	data	parity						
	1	1	0	1	0	1	1	0

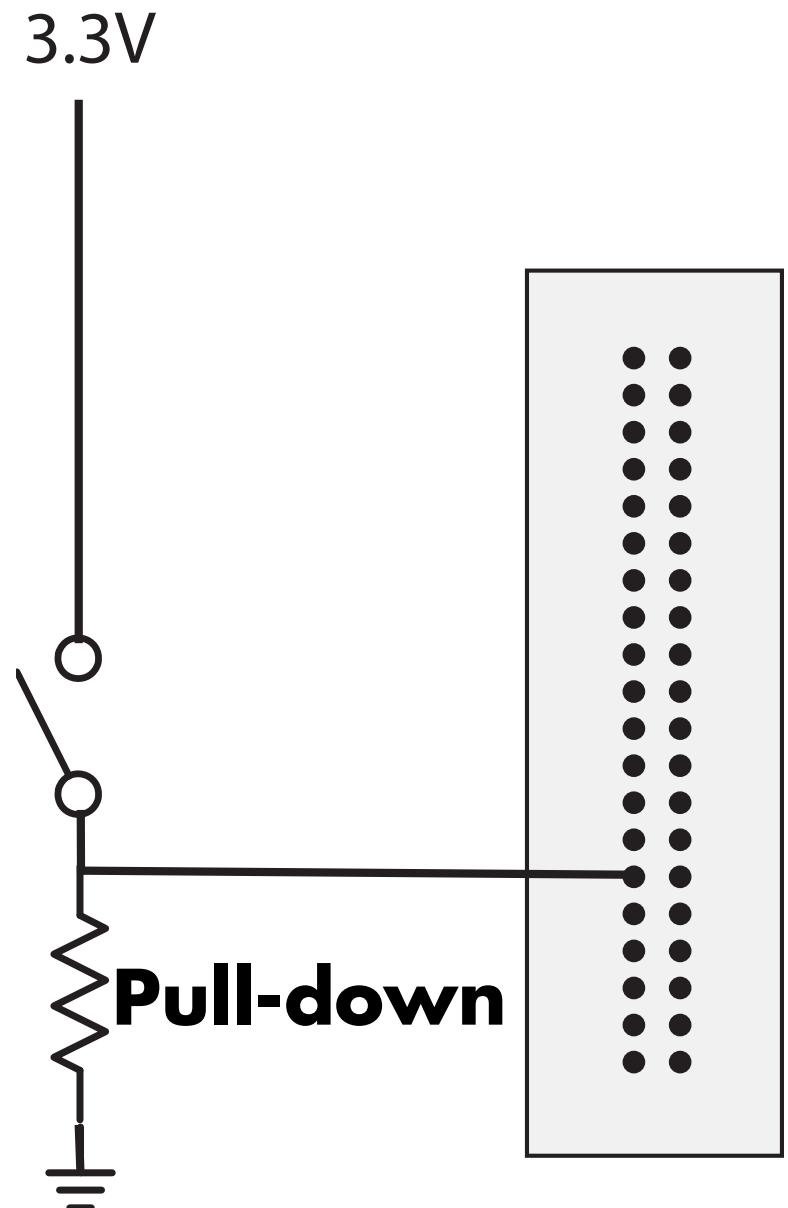
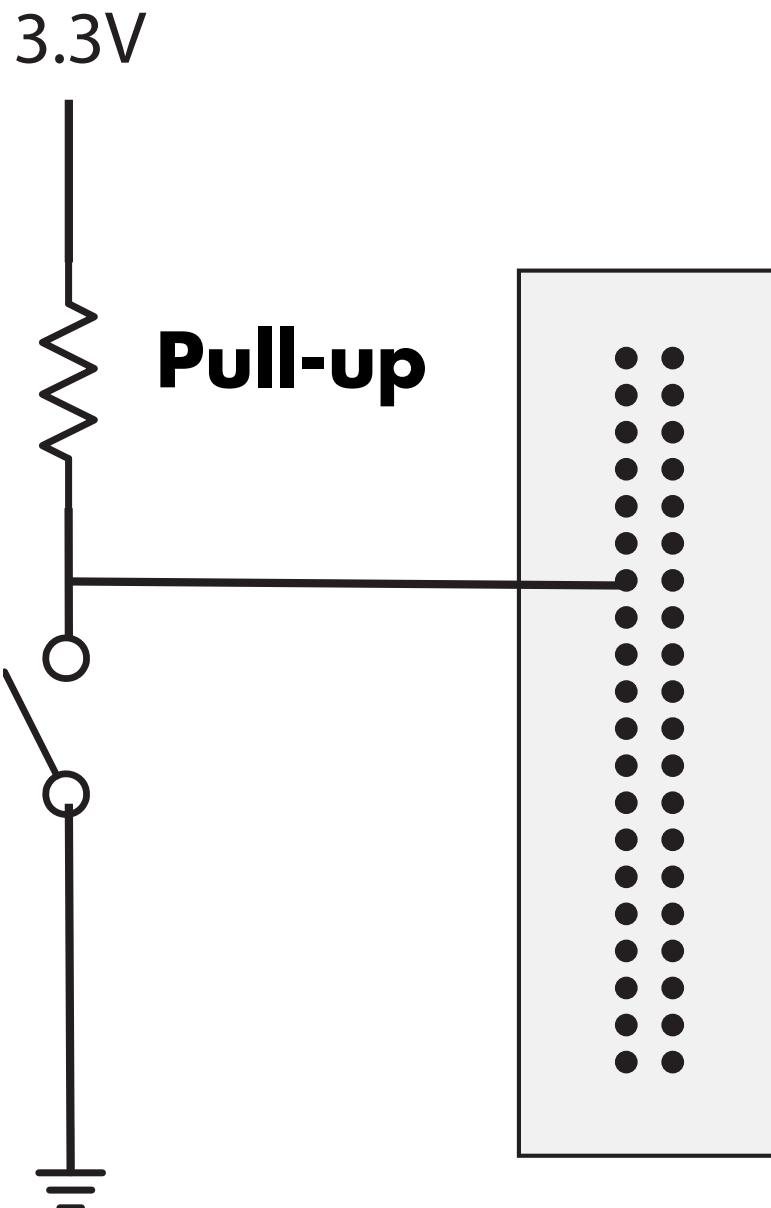
Error in transmission

- Parity is not correct (one bit was flipped)
- Similar to checksum in boot loader

Switch

3.3V





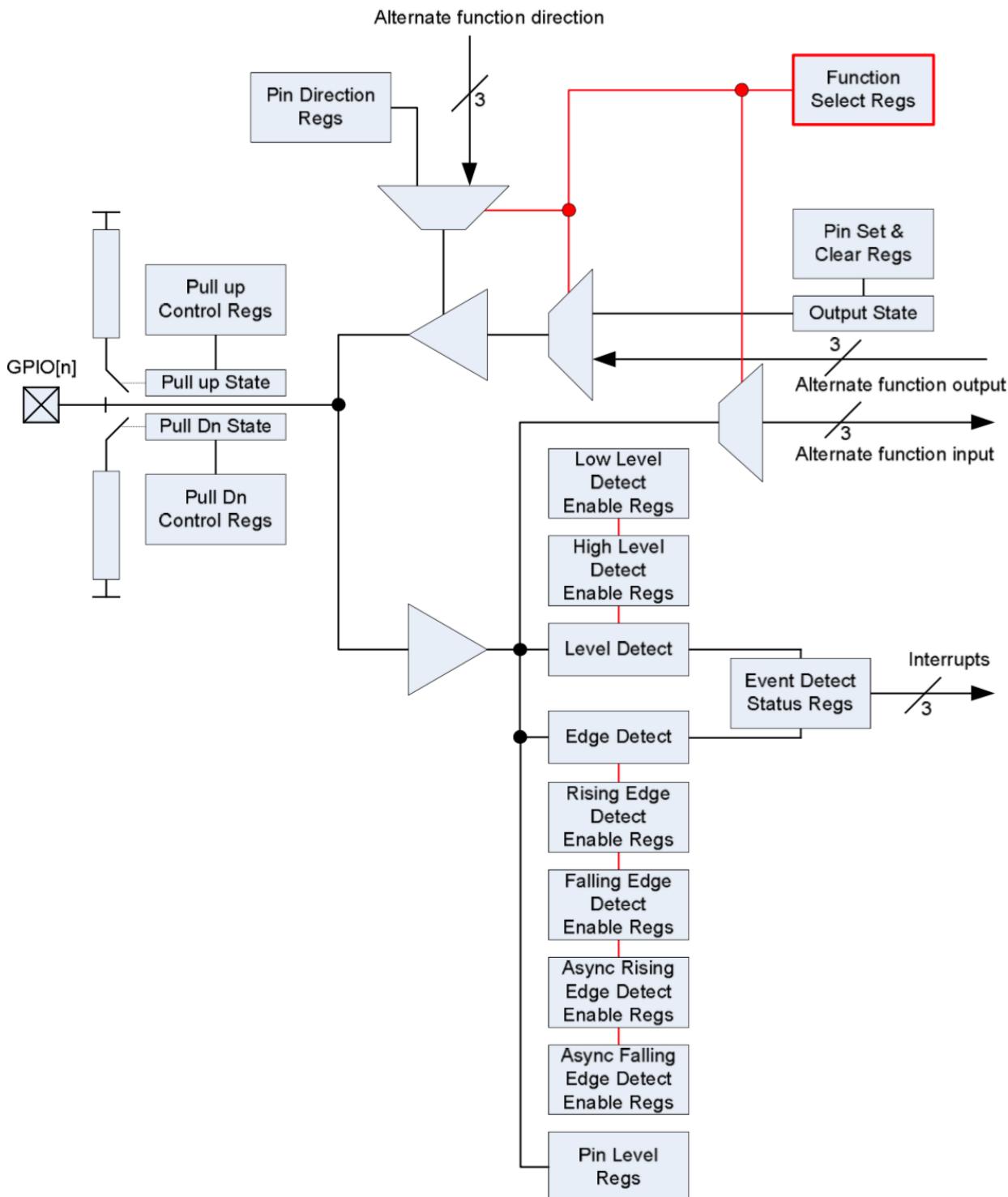
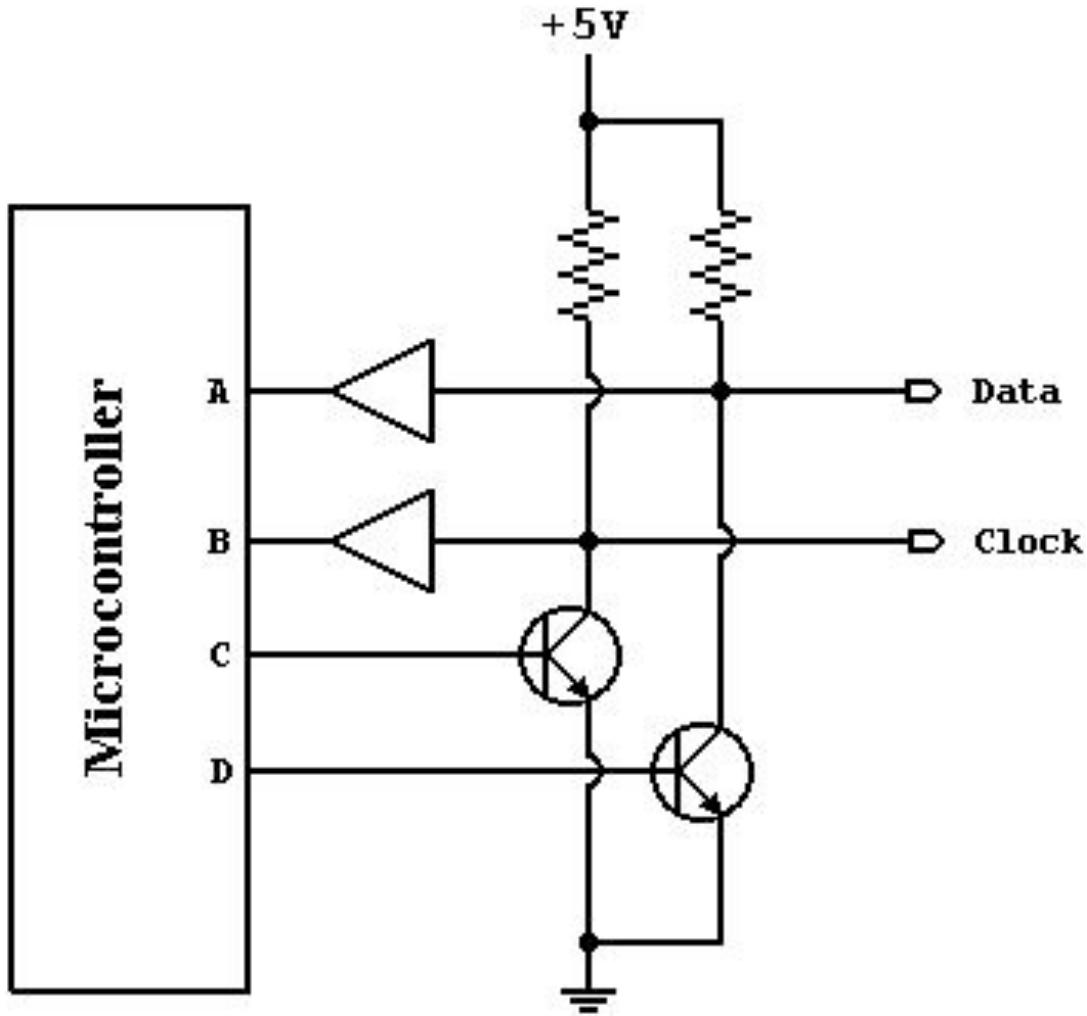


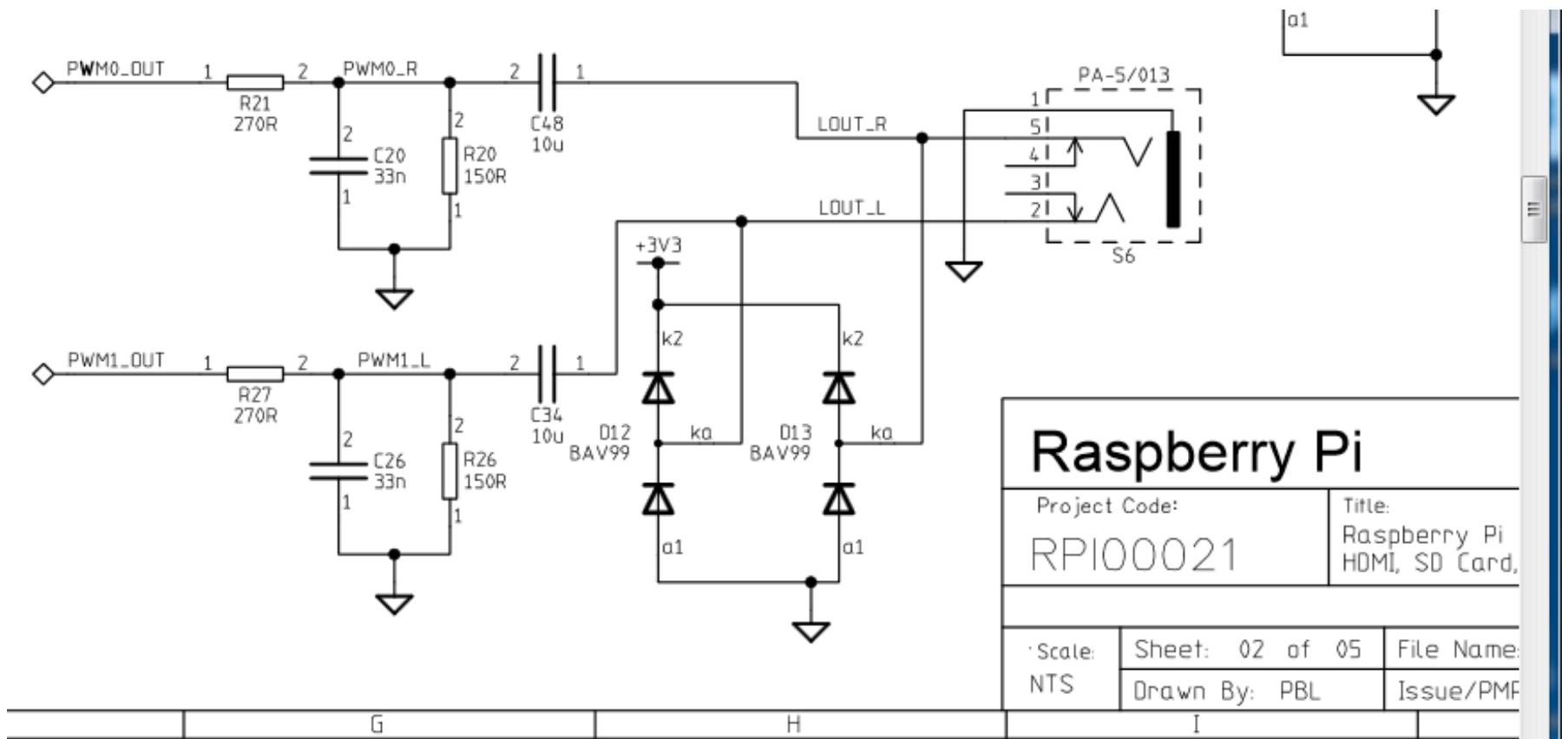
Figure 6-1 GPIO Block Diagram



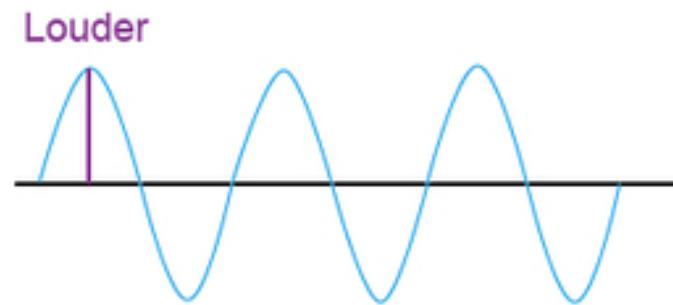
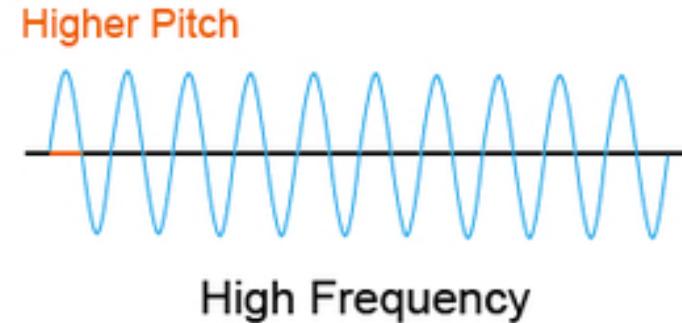
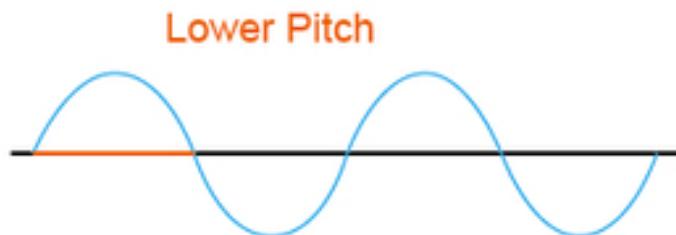
- **DATA and CLK lines are pulled up to 5V**
- **Switching on the transistor sets line to 0V**
- **Enables bi-directional communication (keyboard or Pi can provide data and clock)**

PWM & Sound

Raspberry Pi Stereo Jack



Sound Waves



Pulse-Width Modulation (PWM)

50% duty cycle



75% duty cycle



25% duty cycle



`pwm_clock, pwm_range, pwm_width`

`pwm.c`

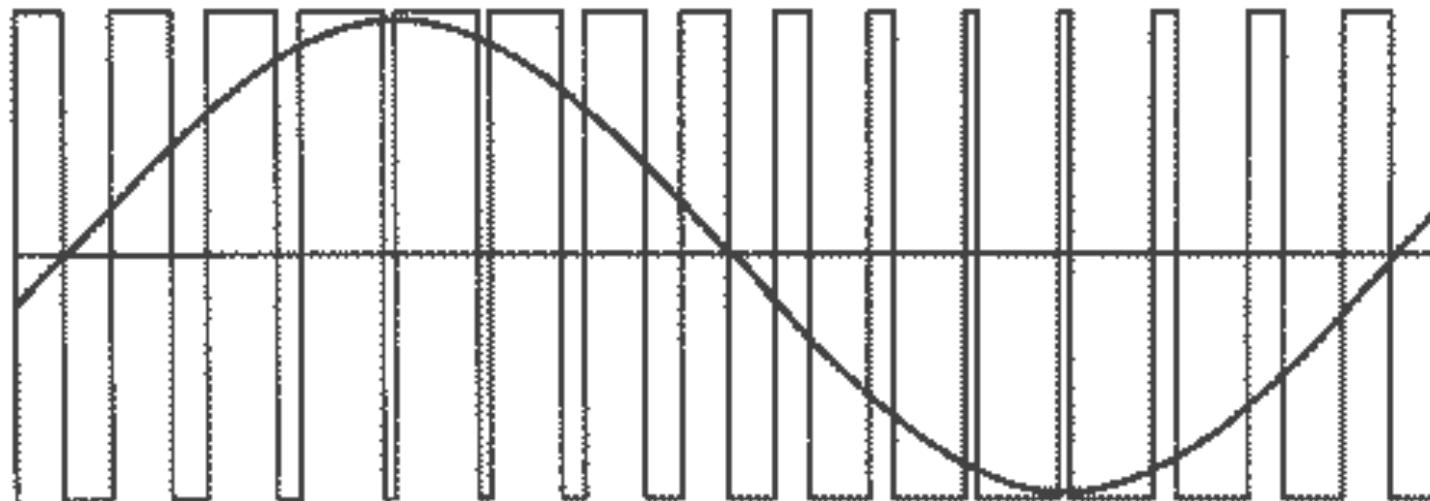
	PWM0	PWM1
GPIO 12	Alt Fun 0	-
GPIO 13	-	Alt Fun 0
GPIO 18	Alt Fun 5	-
GPIO 19	-	Alt Fun 5
GPIO 40	Alt Fun 0	-
GPIO 41	-	Alt Fun 0
GPIO 45	-	Alt Fun 0
GPIO 52	Alt Fun 1	-
GPIO 53	-	Alt Fun 1

**Stereo Jack connected to
GPIO_PIN40 and GPIO_PIN45**

tone.c
melody.c
audio.c

Continuous Values

Can simulate continuous values with fast enough PWM clocking



Like you did to control the LED brightness