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
shell
fb
gl
console

Serial vs. Parallel

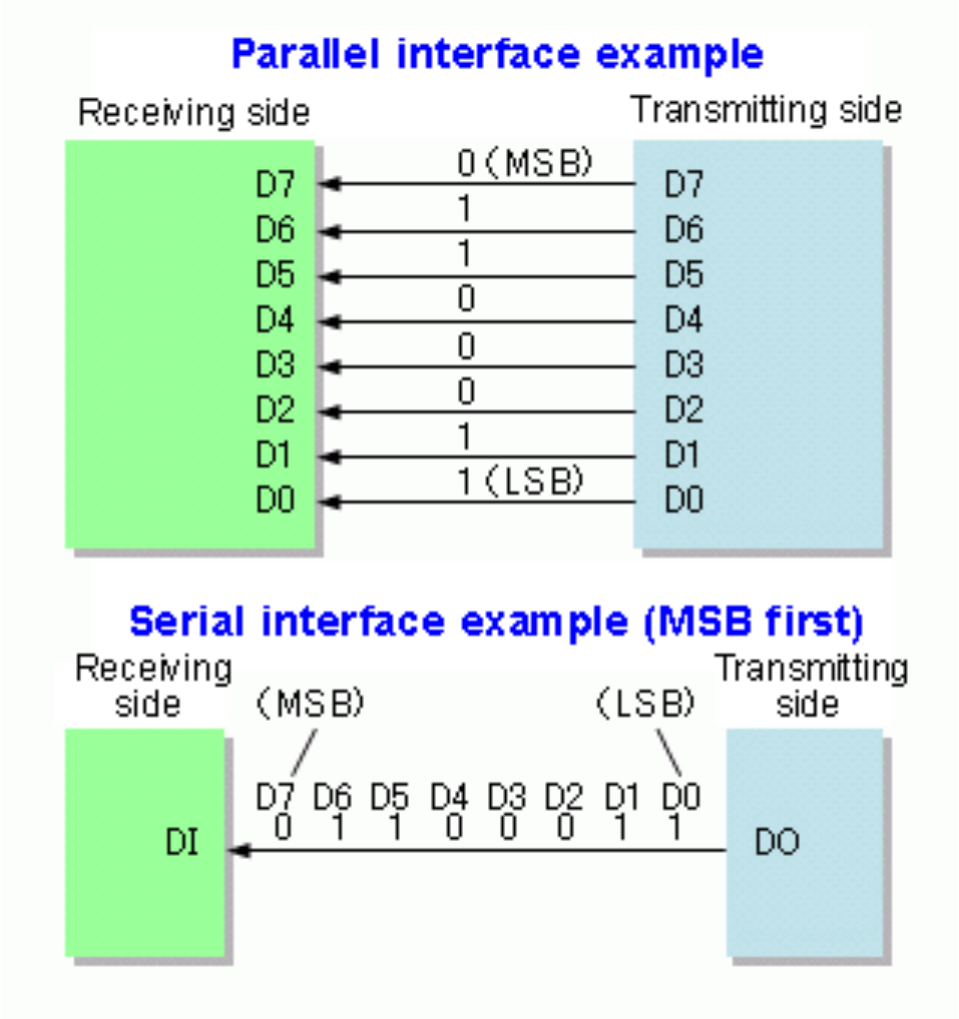


Figure from https://en.wikipedia.org/wiki/Parallel_communication#/media/File:Parallel_and_Serial_Transmission.gif

UART

- **Used in `printf` & the `bootloader`**
- **Start bit, 8 data bits, stop bits**
- **No clock, requires precise timing**

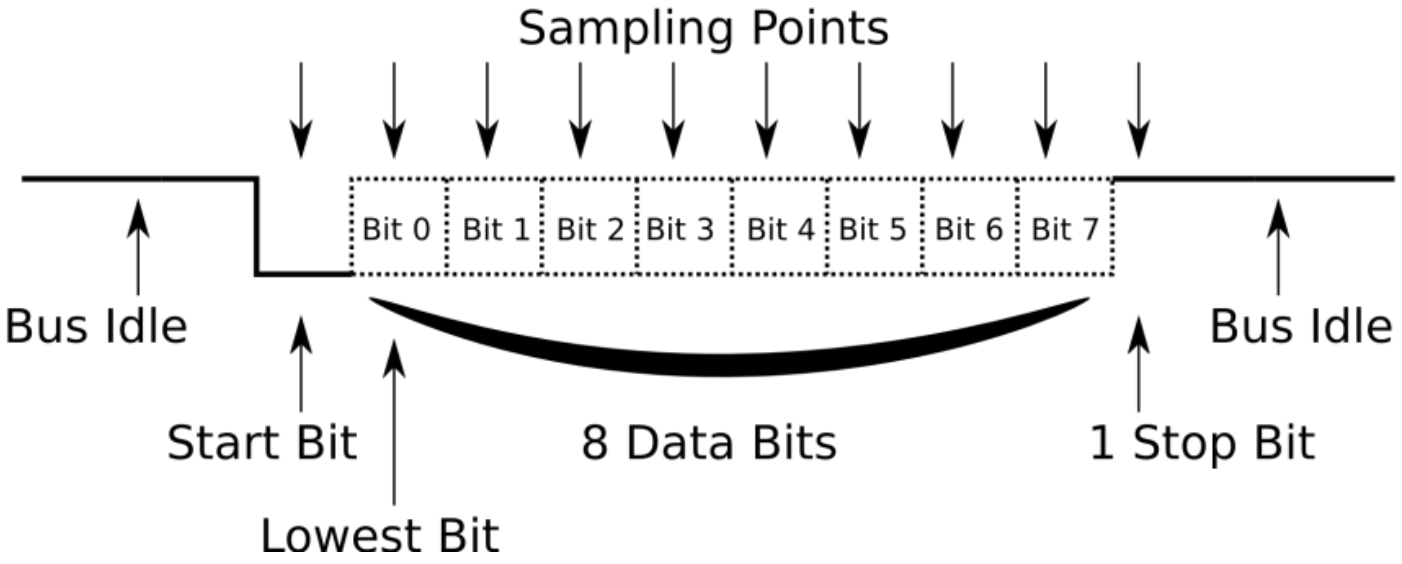
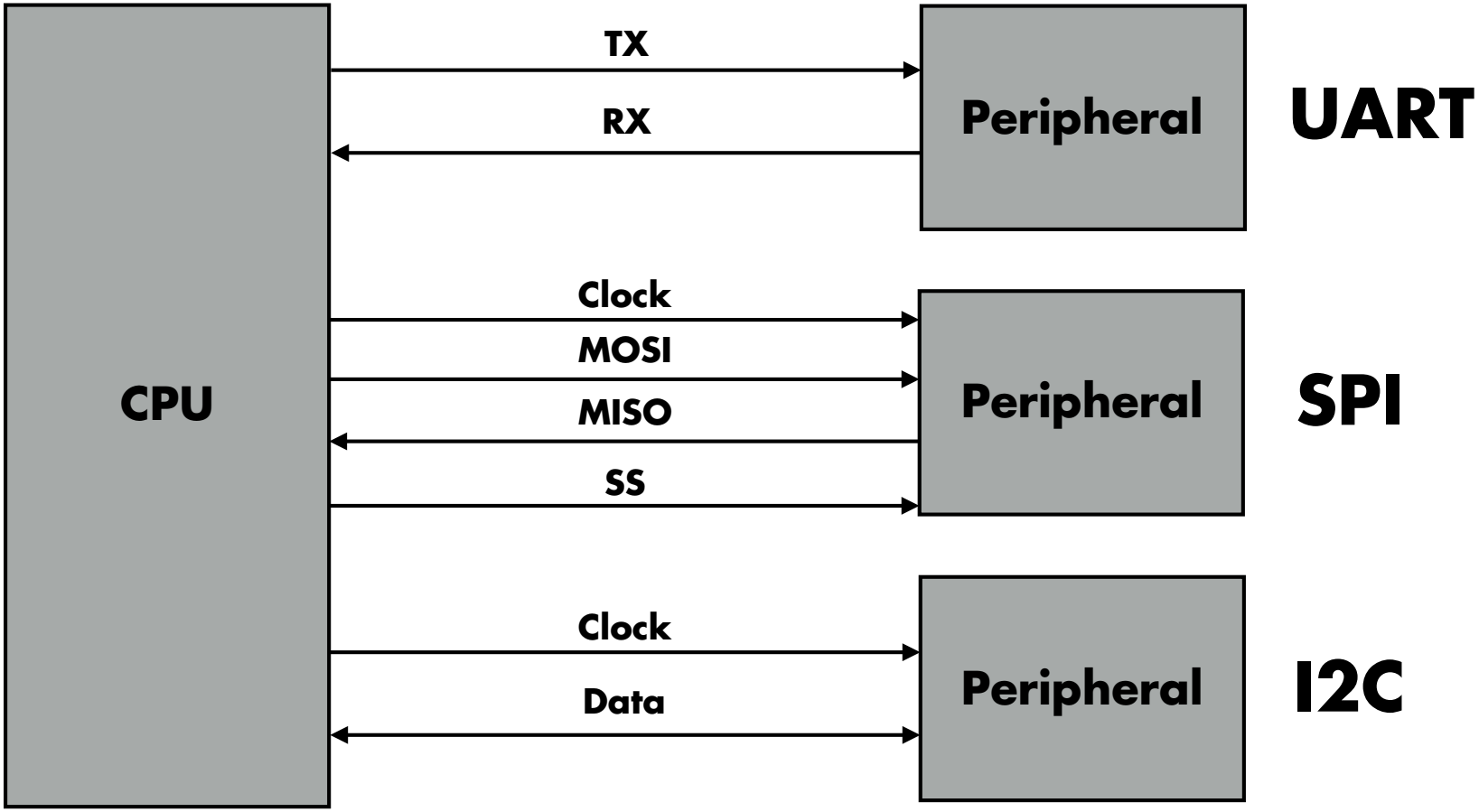


Figure from http://old.myhdl.org/lib/exe/detail.php/projects:rs232image.png?id=projects%3Auart_rs232_receiver_transmitter

Synchronous vs Asynchronous Serial Protocols



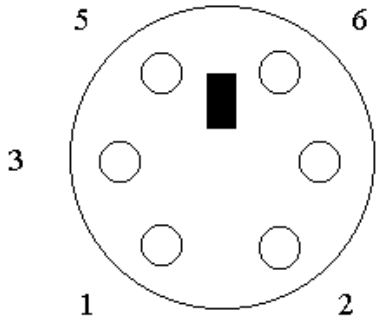
Synchronous has a clock

PS/2 Keyboard

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

Synchronous: CLK and DATA lines

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

6-pin mini-DIN connector

PS/2 Signals Demo

(logic analyzer)

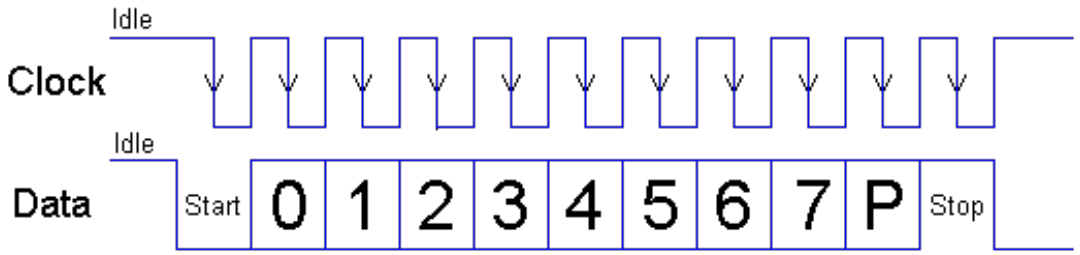
PS/2 Protocol

Keyboard sends clock

Data changes when clock line is high

Host reads data when clock is low

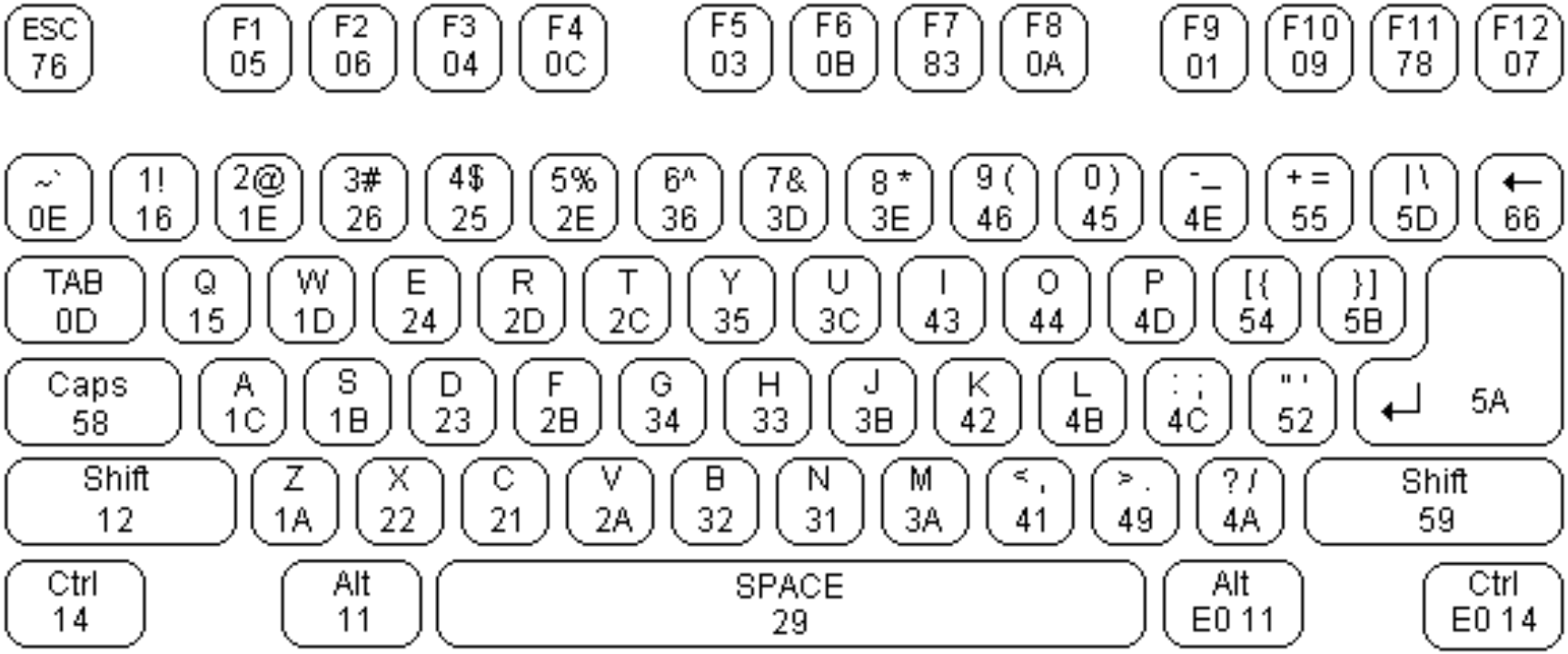
Payload: start bit, 8 data bits (lsb-first), parity bit, 1 stop bit (11 total)



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

Keyboard Scan Codes

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



Make (press) and Break (release) codes

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

Parity Bits

Parity = XOR of data bits

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

even

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

odd

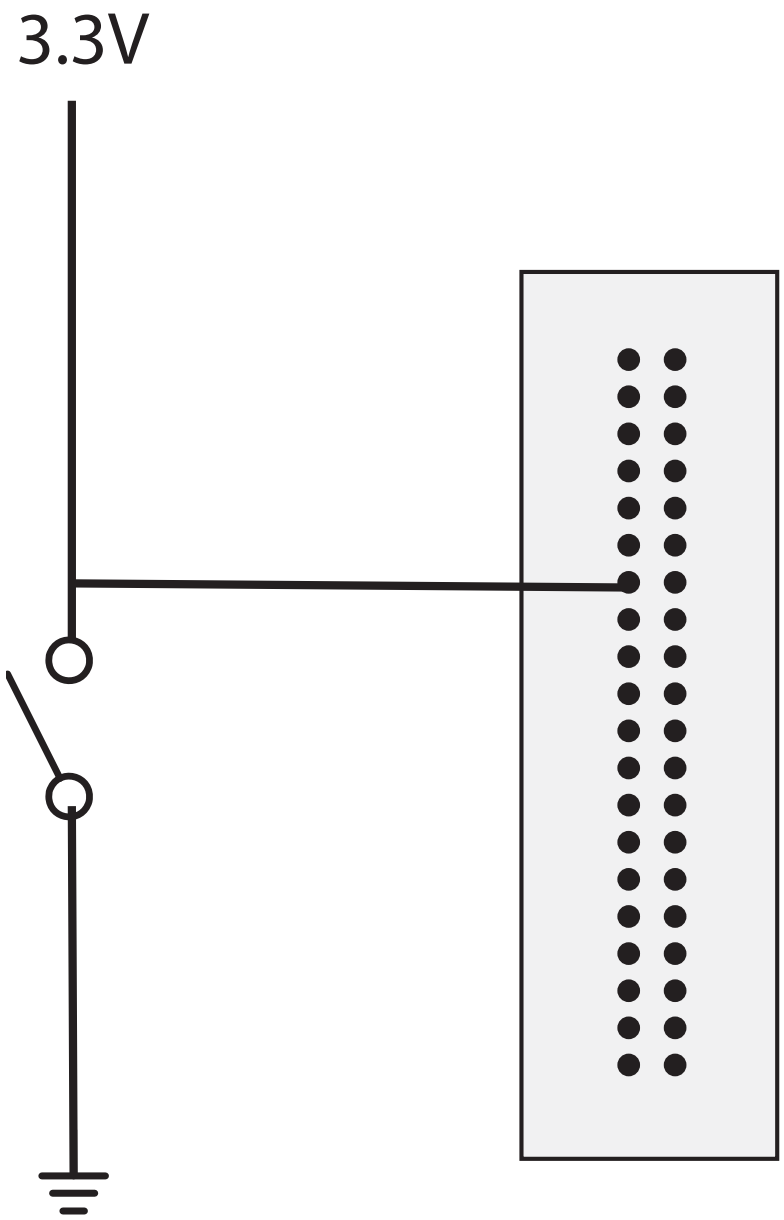
data	data	data	data	data	data	data	data	parity
1	1	0	1	0	1	1	0	0

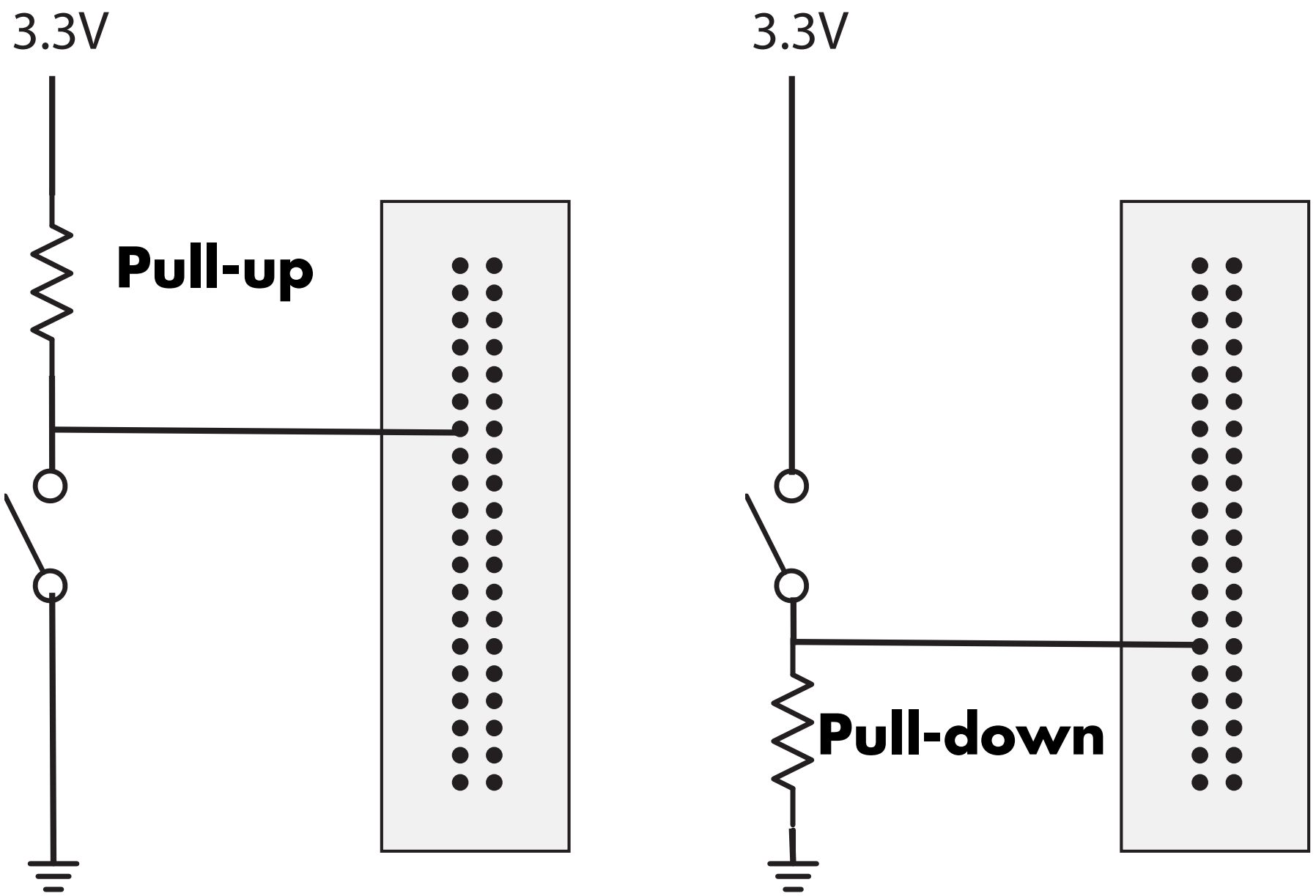
Error in transmission

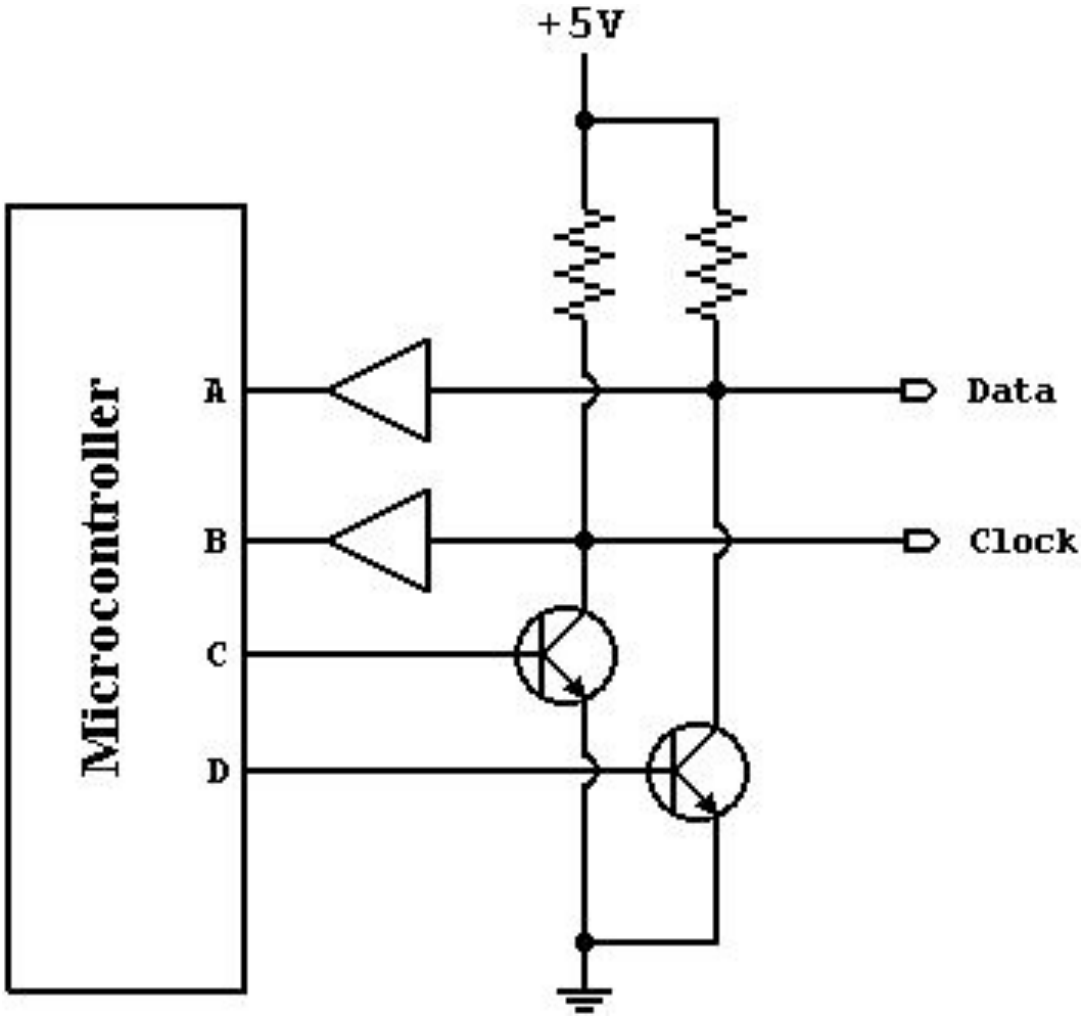
- **Parity is not correct (one bit was flipped)**
- **Similar to checksum in boot loader**

PS2 protocol is odd parity (parity bit makes the number of bits odd)

Switch




3.3V
Pull-up
3.3V
Pull-down



- **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)**

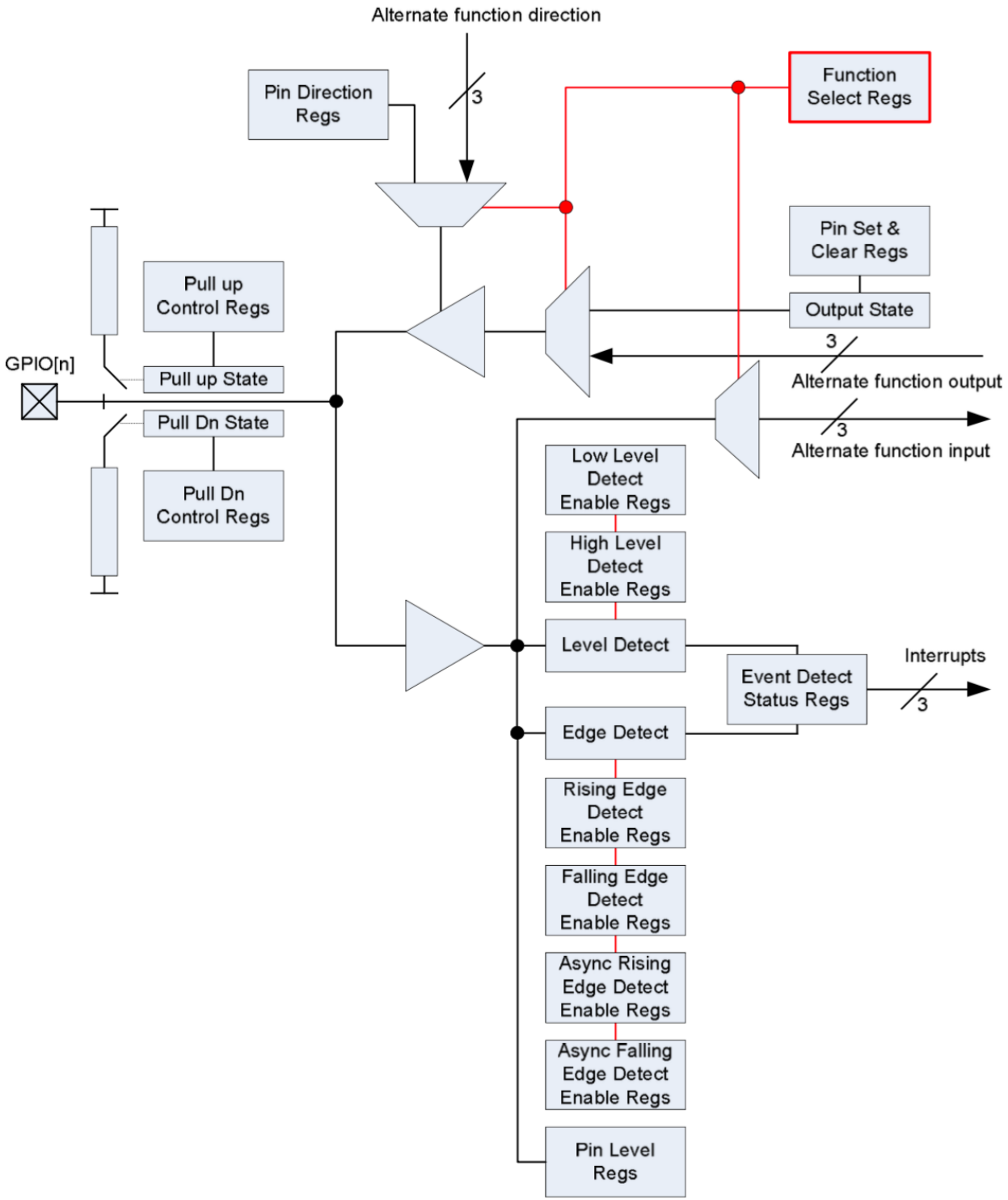
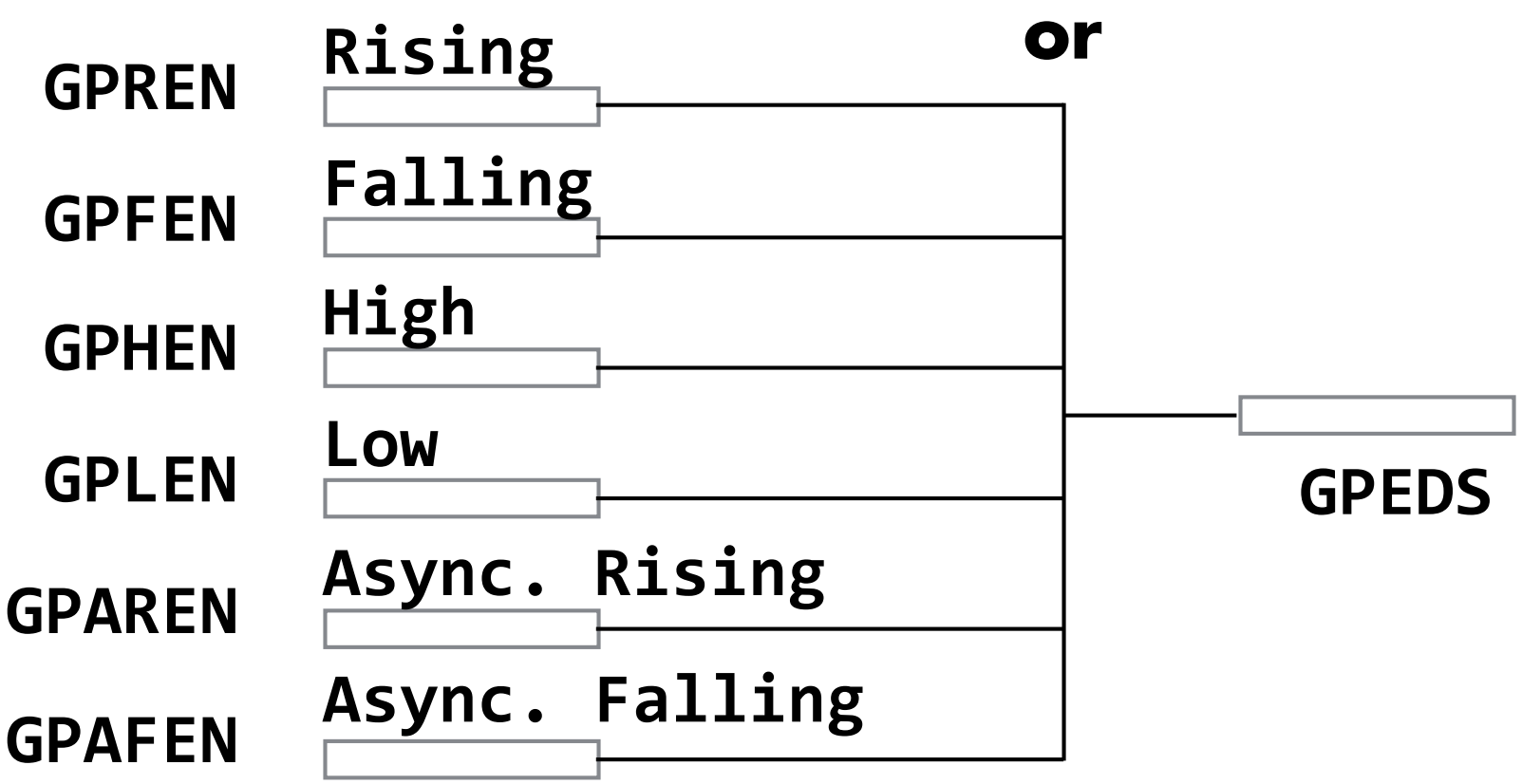


Figure 6-1 GPIO Block Diagram

Reading PS2 Protocol

`ps2/`
`gpioevents/`

GPIO Event Detection



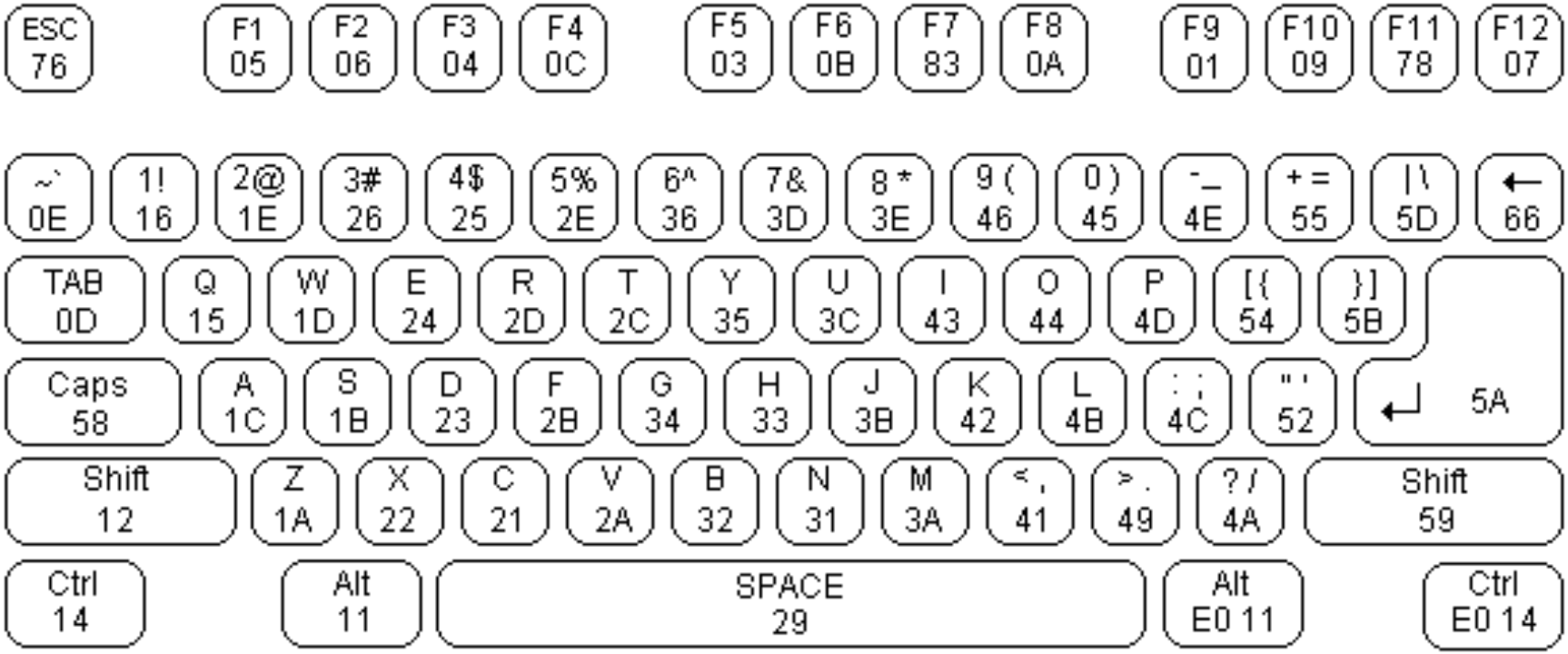
Keyboard Scan Code Demo

`scancode/`


Esc
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
Print Screen SysRq
Scroll Lock
Pause Break
Insert
Home
Page Up
Delete
End
Page Down
Num Lock
Tab
Caps Lock
Enter
Shift
Ctrl
Start
Alt
inland
Pg Up
Pg Dn
Ins
Del

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

```
% man ascii
…
     00 nul   01 soh   02 stx   03 etx   04 eot   05 enq   06 ack   07 bel
     08 bs    09 ht    0a nl    0b vt    0c np    0d cr    0e so    0f si
     10 dle   11 dc1   12 dc2   13 dc3   14 dc4   15 nak   16 syn   17 etb
     18 can   19 em    1a sub   1b esc   1c fs    1d gs    1e rs    1f us
     20 sp    21  !    22  "    23  #    24  $    25  %    26  &    27  '
     28  (    29  )    2a  *    2b  +    2c  ,    2d  -    2e  .    2f  /
     30  0    31  1    32  2    33  3    34  4    35  5    36  6    37  7
     38  8    39  9    3a  :    3b  ;    3c  <    3d  =    3e  >    3f  ?
     40  @    41  A    42  B    43  C    44  D    45  E    46  F    47  G
     48  H    49  I    4a  J    4b  K    4c  L    4d  M    4e  N    4f  O
     50  P    51  Q    52  R    53  S    54  T    55  U    56  V    57  W
     58  X    59  Y    5a  Z    5b  [    5c  \    5d  ]    5e  ^    5f  _
     60  `    61  a    62  b    63  c    64  d    65  e    66  f    67  g
     68  h    69  i    6a  j    6b  k    6c  l    6d  m    6e  n    6f  o
     70  p    71  q    72  r    73  s    74  t    75  u    76  v    77  w
     78  x    79  y    7a  z    7b  {    7c  |    7d  }    7e  ~    7f del
…
```

Keys (Scan Codes) ≠ Characters (ASCII)

Scan code numbers ≠ ASCII character codes

- Typically 104 keys
- 127 ASCII character codes

Extra keys

- Special keys - interpreted by the OS or App
  - F1, ..., F12
  - Arrows, insert, delete, home, ...
- Multiple keys with same function
  - Left and right shift, ...
  - Numbers on keypad vs. keyboard

Triggering a Rebuild

Questions?

Suppose you change `gpio.c`, what needs to be rebuilt?

Suppose you change `gpio.h`, what needs to be rebuilt?

Could you need to rebuild if you changed the `Makefile`?

depend/

Keyboard Abstractions

Layers (keyboard.h)

unsigned char keyboard_read_scancode(void)

- **returns scancode**

key_action_t keyboard_read_sequence(void)

- **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**

Keys ≠ Characters

Keyboard Viewer

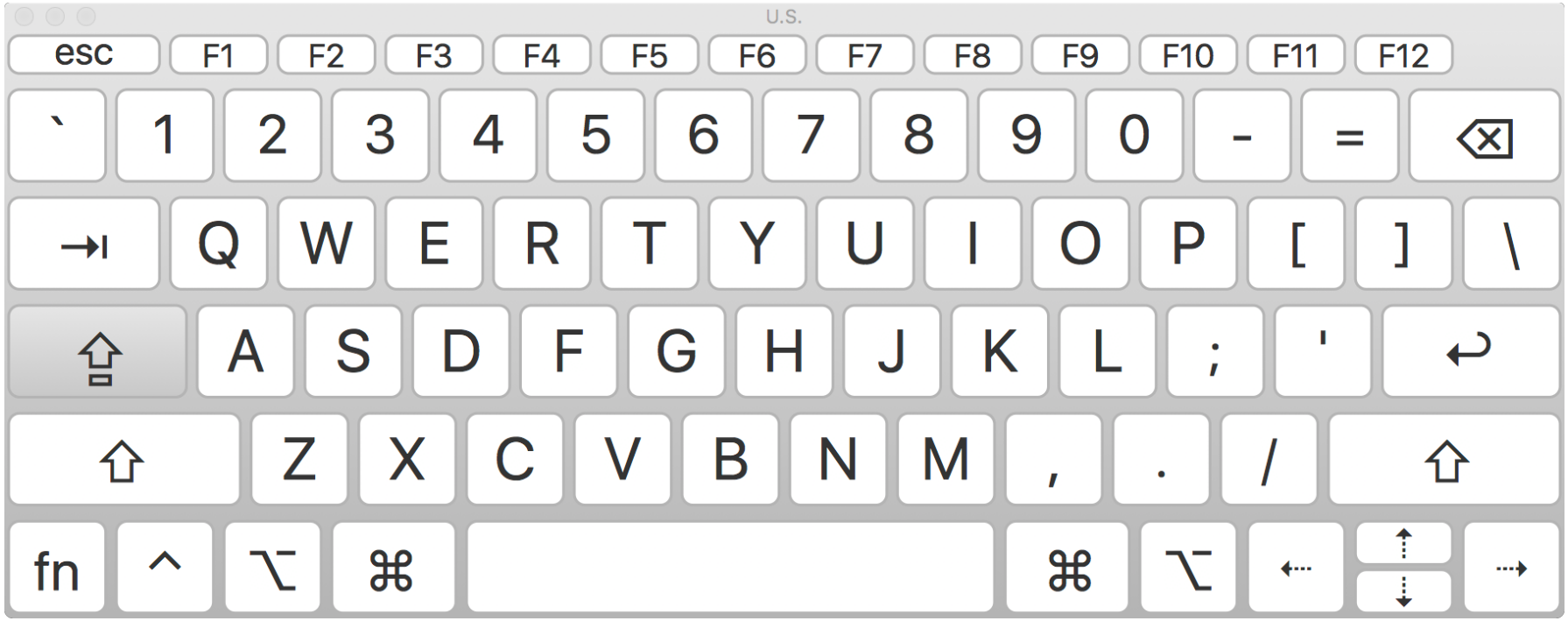
Keys ≠ Characters

Modifier keys

[Shift]

Keys ≠ Characters

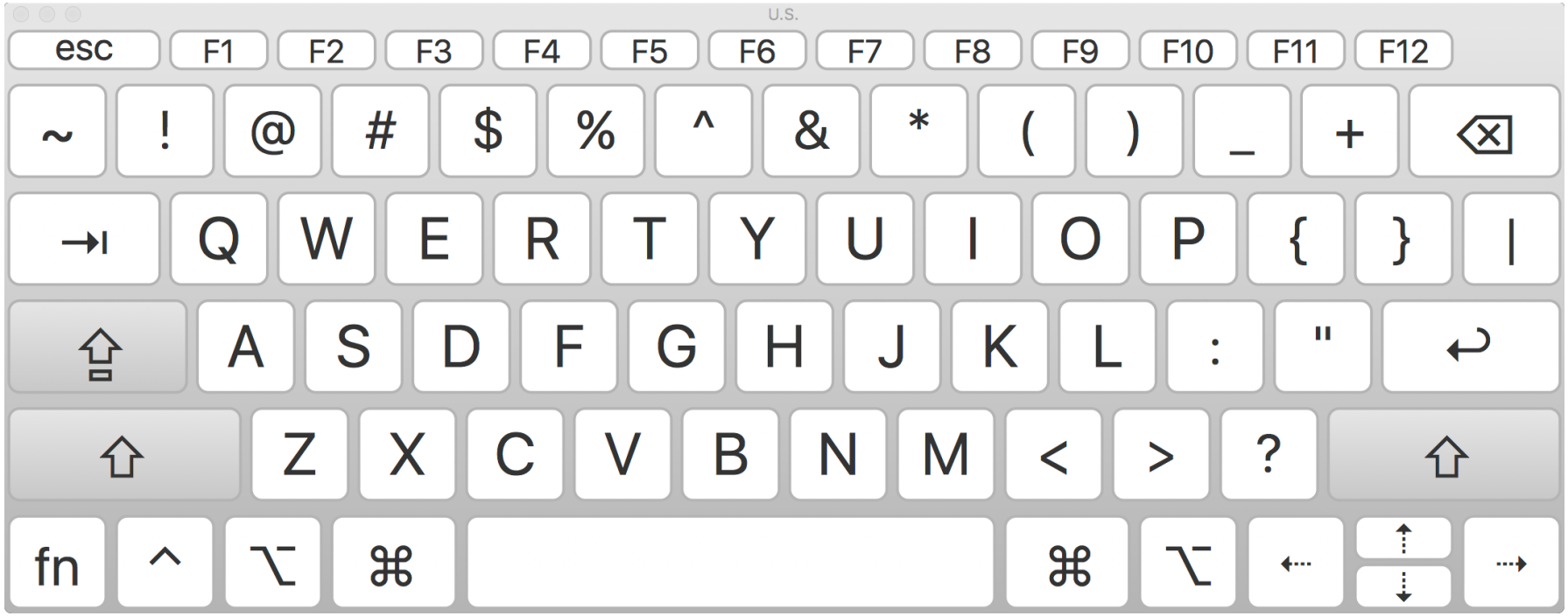
Modifier keys



[Caps Lock]

Keys ≠ Characters

Modifier keys



[Shift + Caps Lock]

Keys ≠ Characters

Modifier keys

[Option]

Orange = Dead Keys

Keys ≠ Characters

Modifier keys

[Option-A]

MIDI

MIDI:
Musical Instrument Digital Interface

Simple interface to control musical instruments

Emerged from electronic music and instruments in 1970s

First version described in Keyboard magazine in 1982

MIDI

31.25 kbps 8-N-1 serial protocol

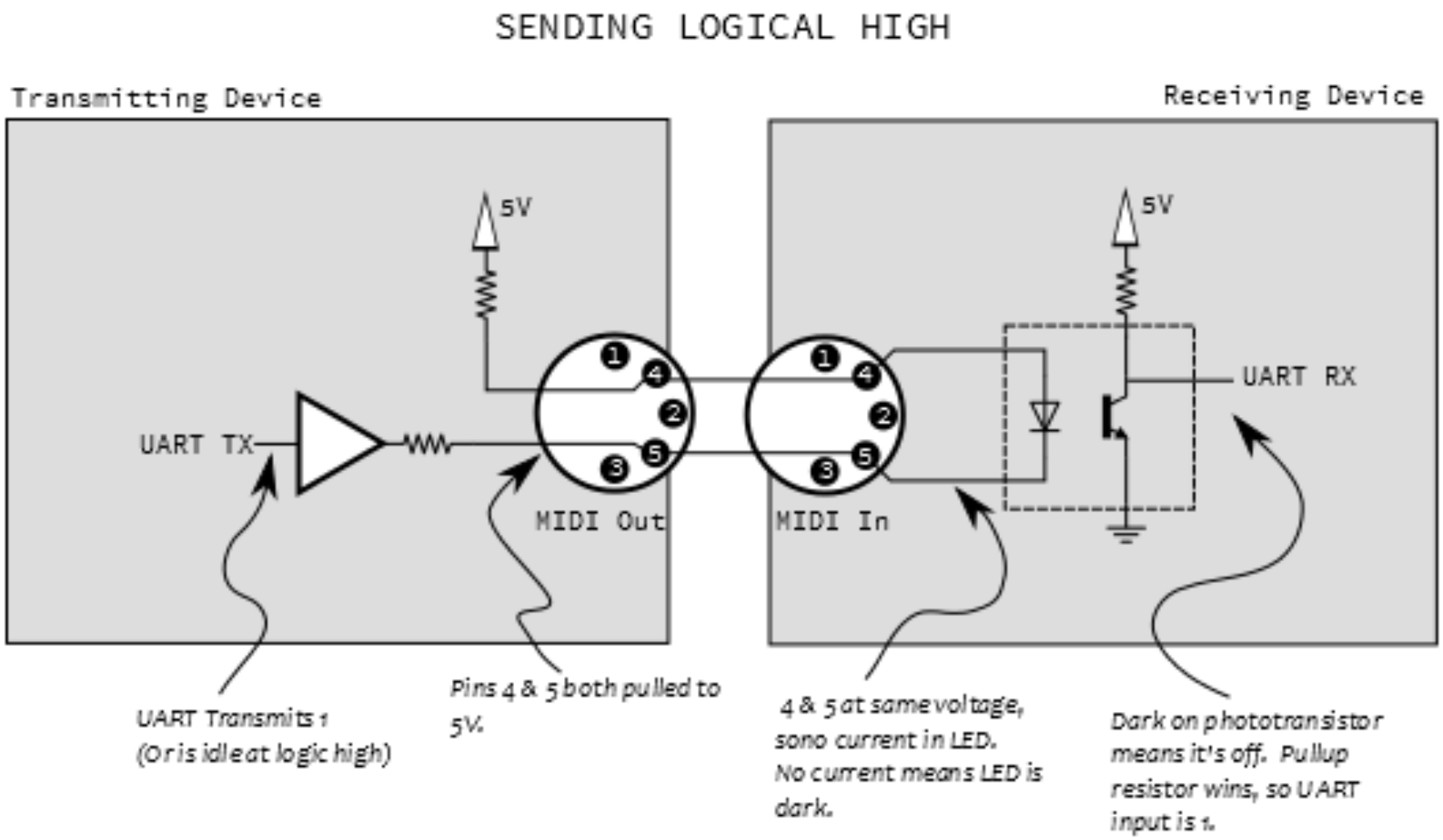
Commands are 1 byte, with variable parameters (c=channel, k=key, v=velocity, l=low bits, m=high bits)

Command	Code	Param	Param
Note on	`1001cccc`	`0kkkkkkk`	`0vvvvvvv`
Note off	`1000cccc`	`0kkkkkkk`	`0vvvvvvv`
Pitch bender	`1110cccc`	`0lllllll`	`0mmmmmmm`

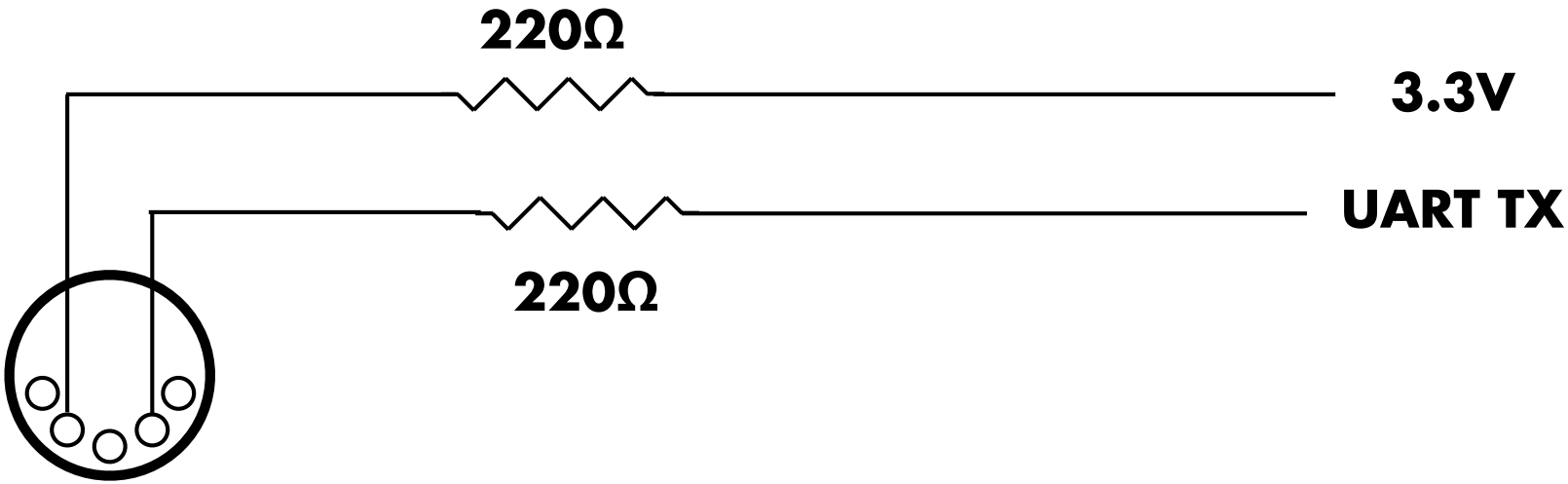
A bit of “music”

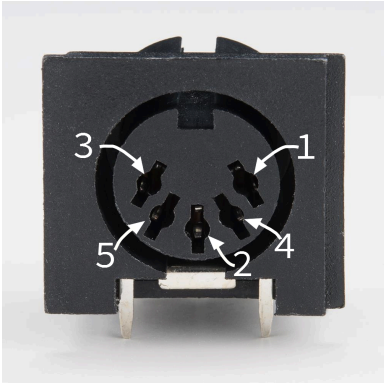
`code/midi/midi-lamb.c`

MIDI Circuit



MIDI Transmit Circuit





MIDI Receive Circuit