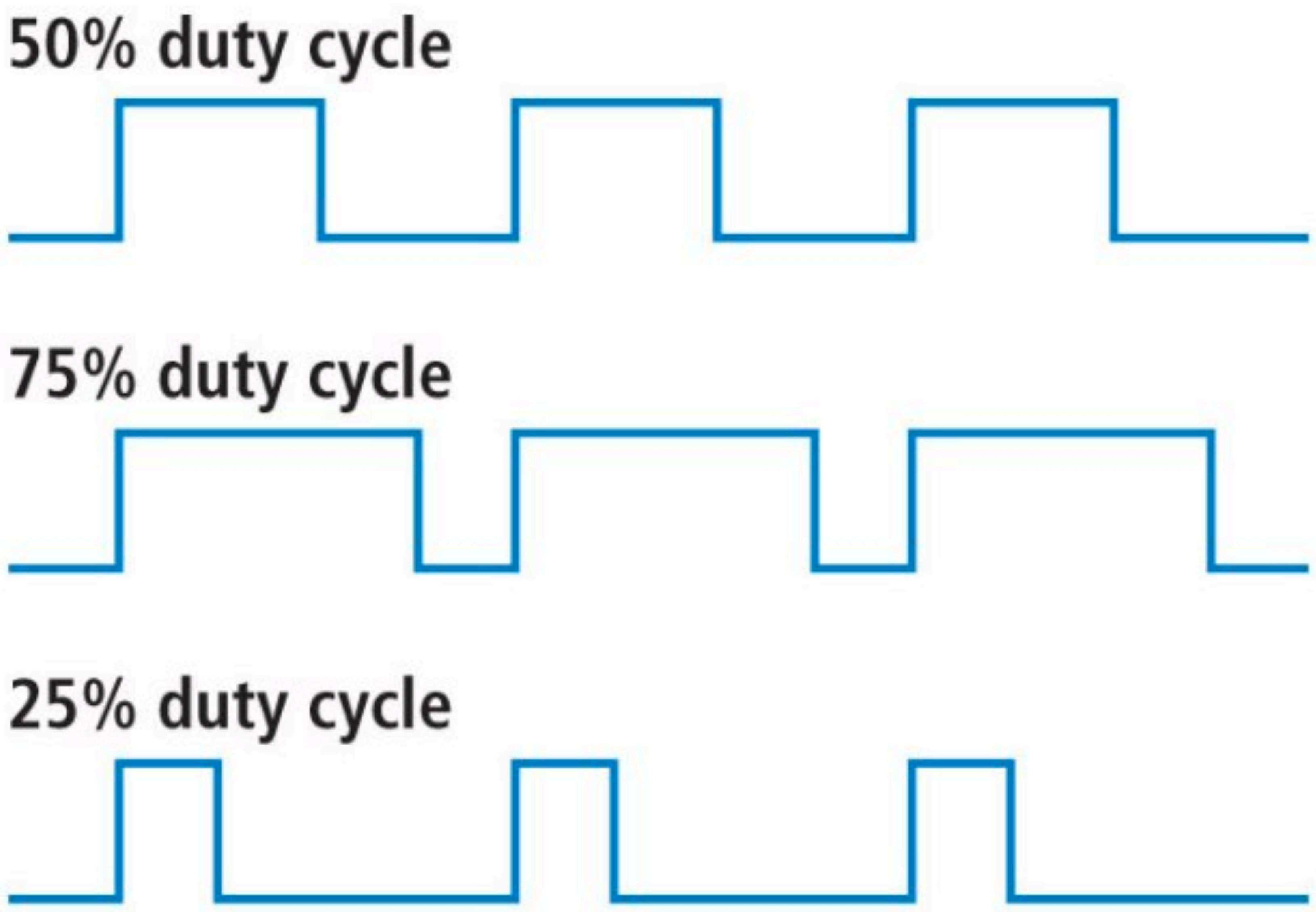
Sound

Pulse-Width Modulation (PWM)



`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

PWM0 is output on GPIO_PIN18 ALT_FUN5

Hardware PWM Support

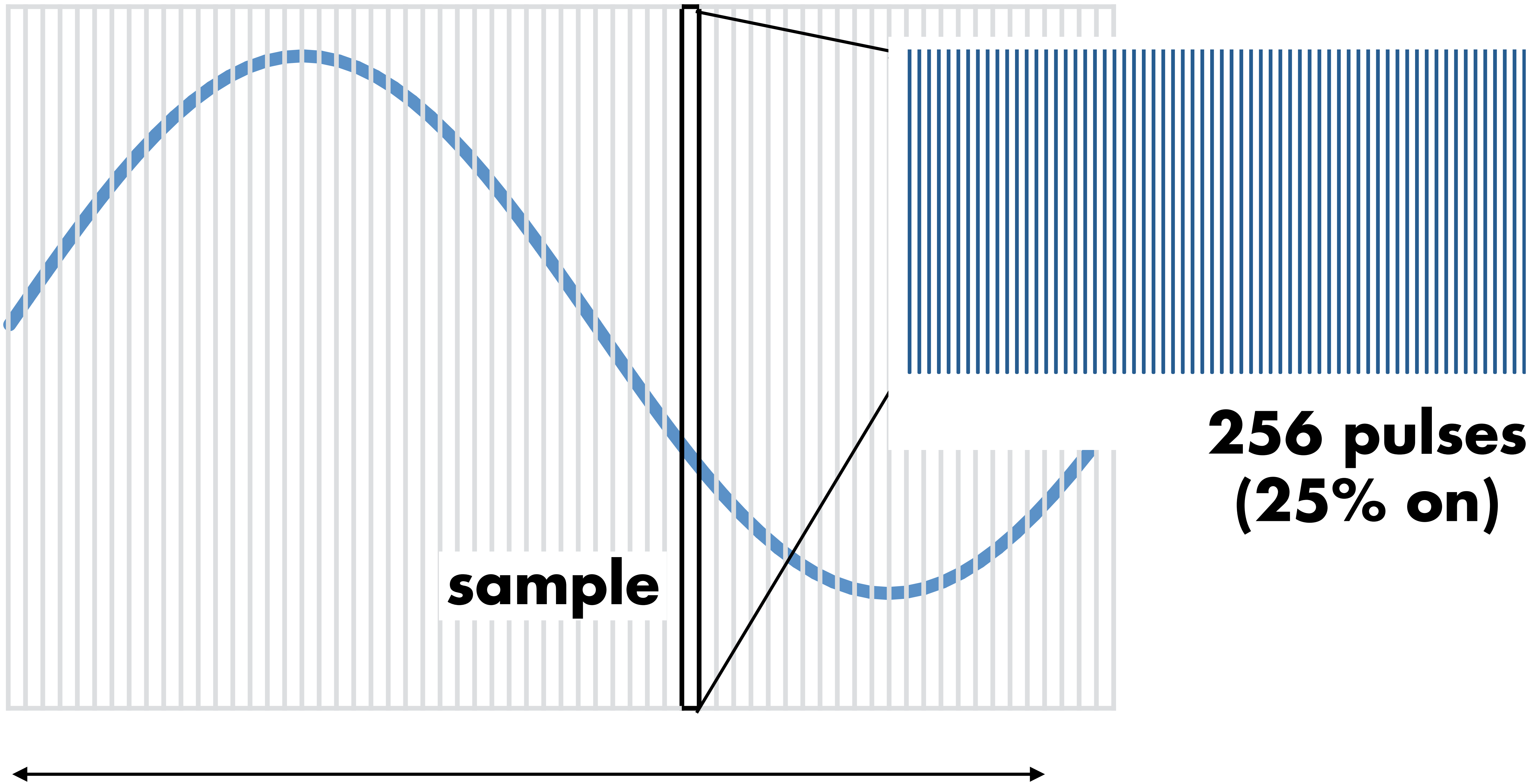
Start with a 19.2MHz clock, divide it to specify the time slots of on/off

E.g., divider of 2.375 = 8,192kHz

Divide wave into steps (e.g., 64)

Divide each step into train of (e.g., 256) pulses: tell hardware how many pulses should be high

Example: Sine

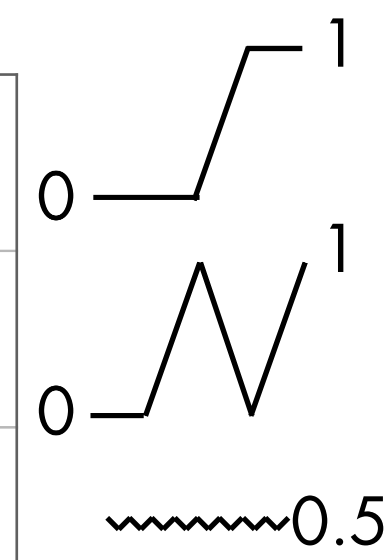


64 samples

1kHz wave * 64 samples * 256 pulses = 8,192kHz

PWM Clocking of Pulses

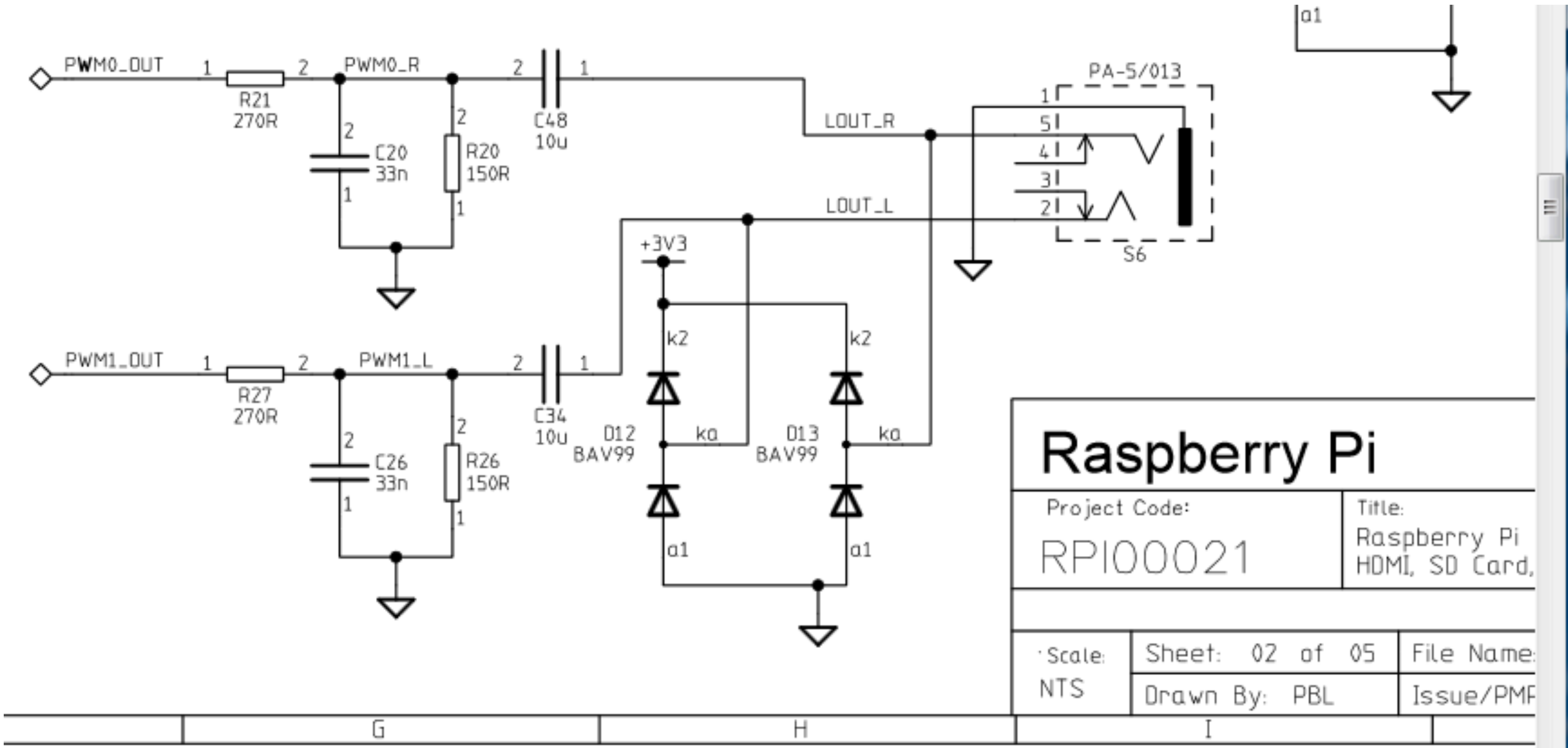
bad	0	0	0	0	0	0	0	0	+V	+V	+V	+V	+V	+V	+V	+V
fair	0	0	0	0	+V	+V	+V	+V	0	0	0	0	+V	+V	+V	+V
good	0	+V	0	+V	0	+V	0	+V	0	+V	0	+V	0	+V	0	+V



step

pwm.c
tone.c
melody.c

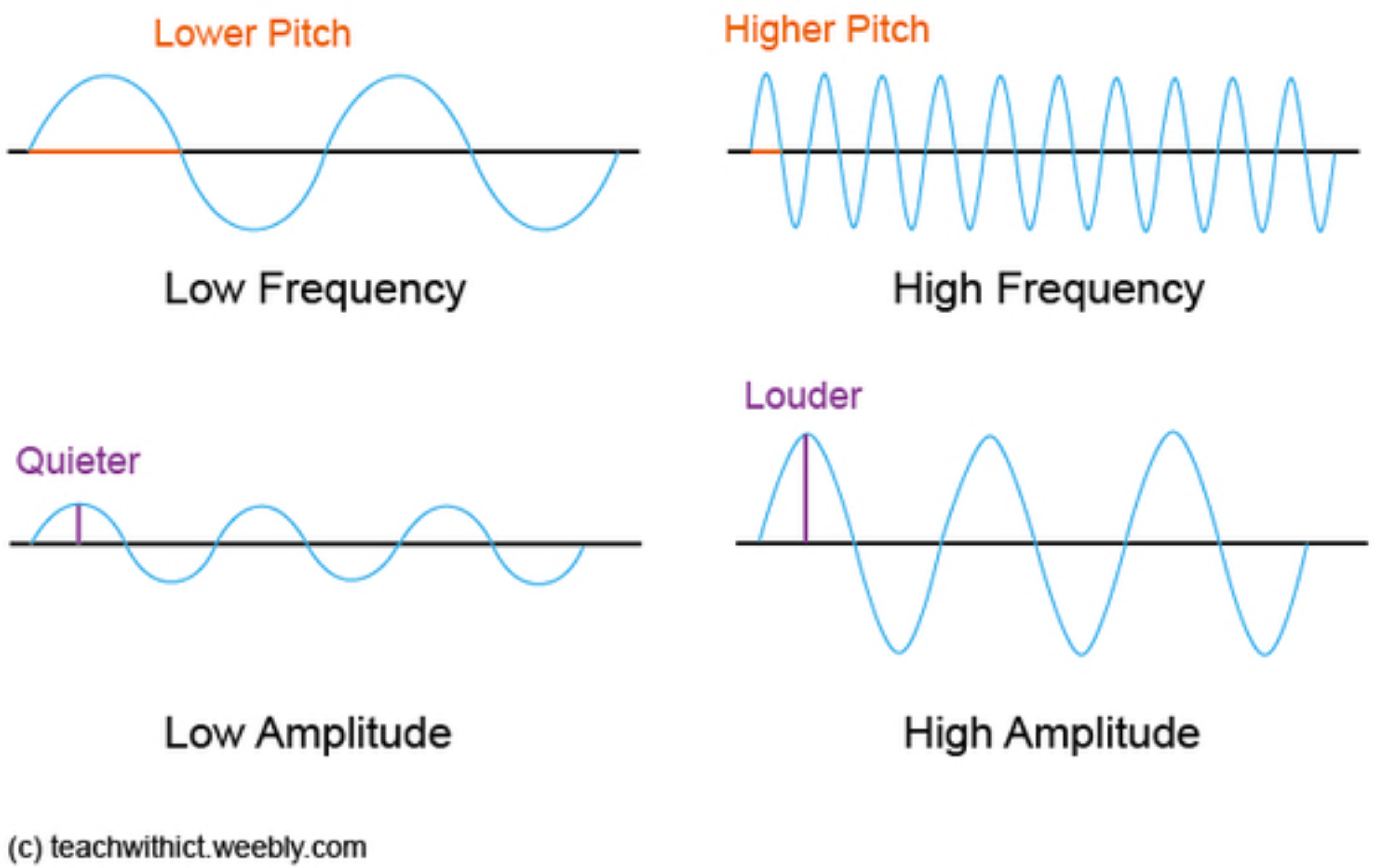
Raspberry Pi Stereo Jack



	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

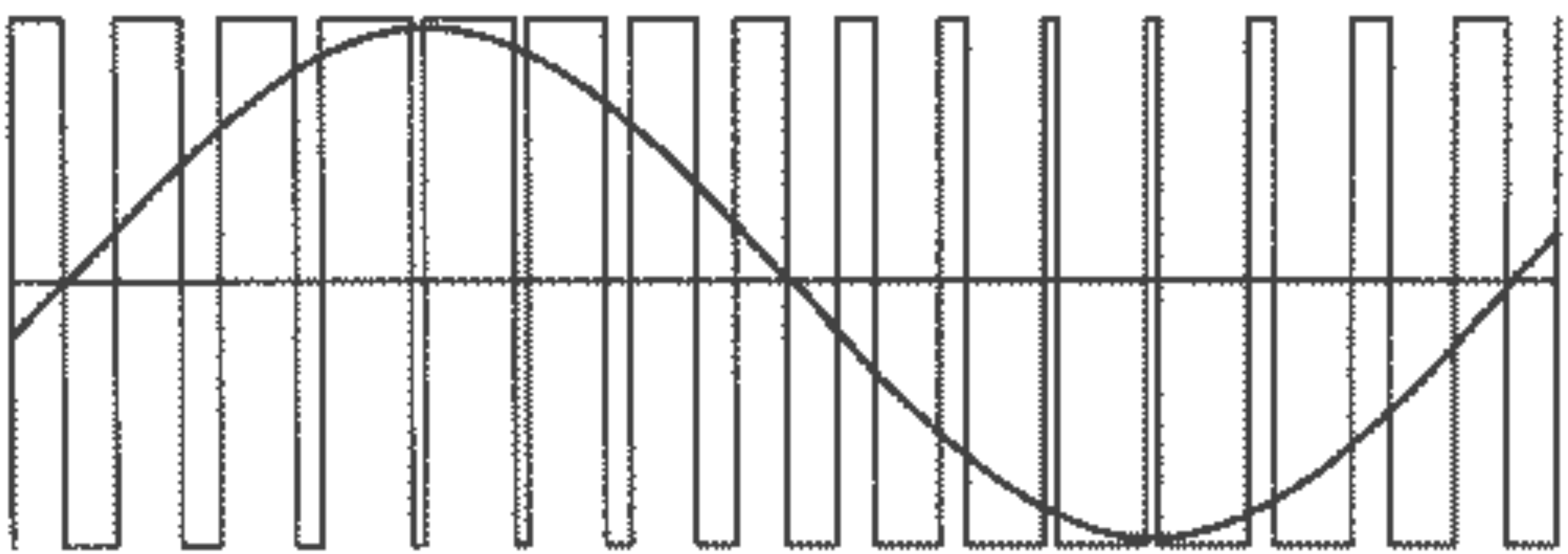
Sound Waves



Lower Pitch
Low Frequency
Higher Pitch
High Frequency
Quieter
Low Amplitude
Louder
High Amplitude
(c) teachwithict.weebly.com

Continuous Values

Can simulate continuous values with fast enough PWM clocking



Like you did to control the LED brightness

audio.c

MIDI

What if we want real music?

MIDI

*Actually, this is kind of fake music

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

A bit of “music”

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`

UART (2+ pins)

Bidirectional data transfer, no clock line — "asynchronous".

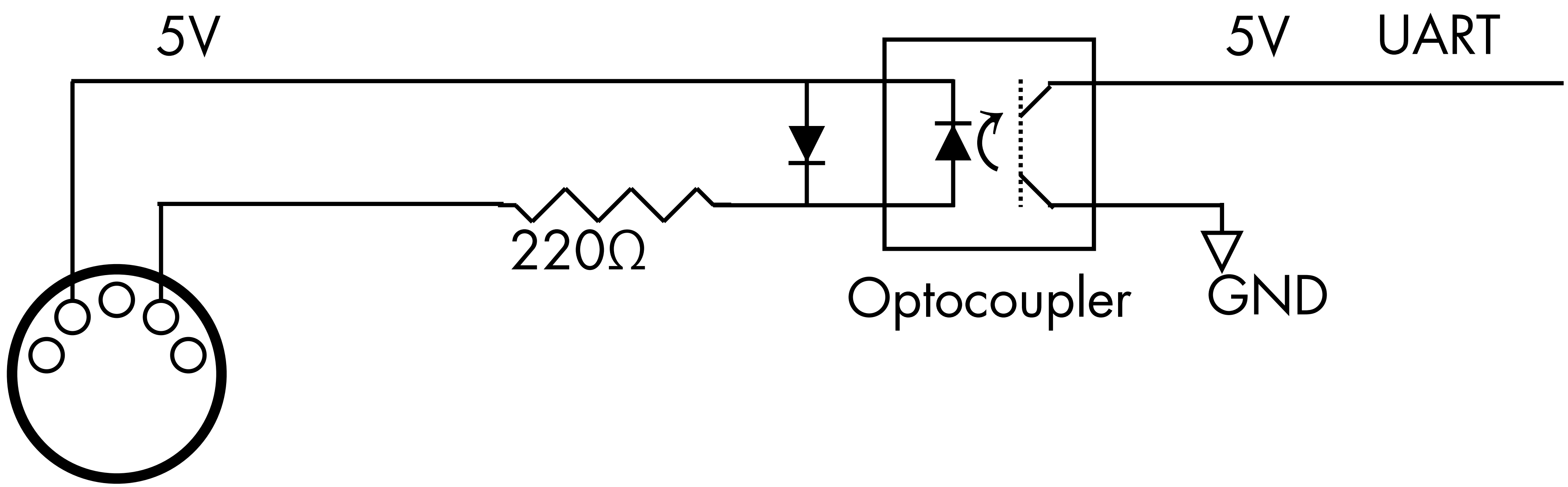
Additional pins for flow control ("I'm ready to send"), old telephony mechanisms.

Start bit, (5 to 9) data bits, (0 or 1) parity bit, (1 or 2) stop bit. 8-N-1:

start	data	data	data	data	data	data	data	data	parity	stop	stop
0	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1		1	1

MIDI Circuit

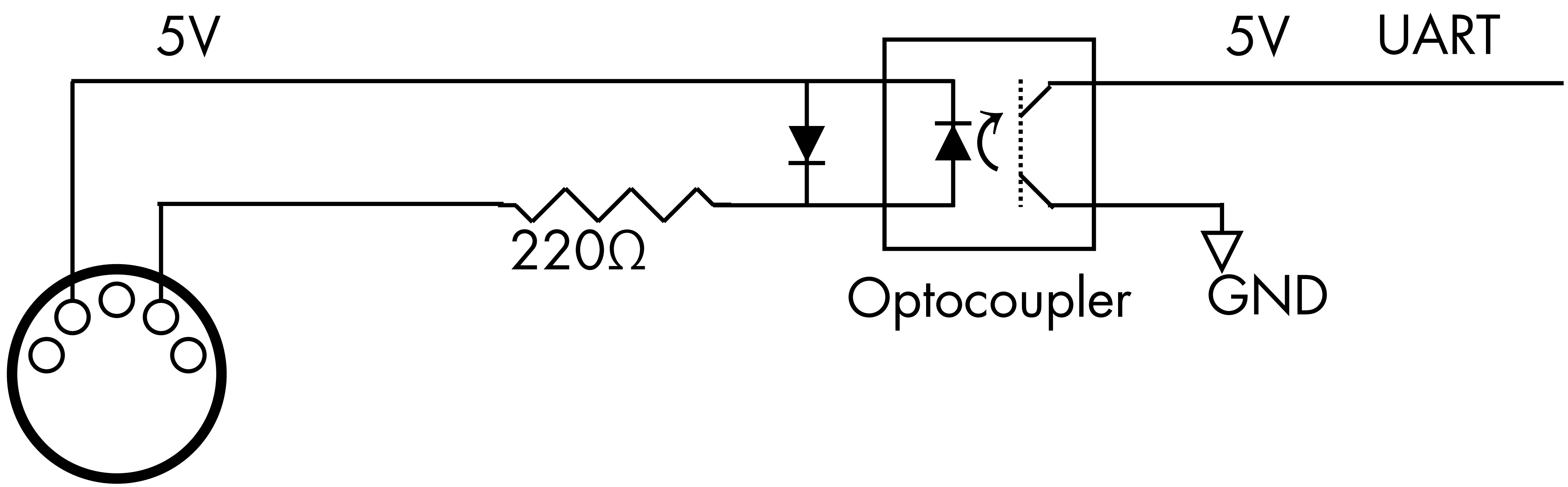
0 is high, 1 is low!



Optocoupler completely isolates circuits electrically: no noise in instrument

MIDI Hack!

If we don't have an optocoupler, we can do okay with an additional 220Ω resistor:



`code/midi`

Raspberry Pi hooked up to a MIDI keyboard on GPIO pin 25.

UART timing

Inversion